The Causal Impact of Information Crowd-sourcing Platform on Agricultural Supply Chain

by

Teuku Mahfuzh Aufar Kari
B.Sc. in Chemistry Nanyang Technological University, 2011
M.Sc. in Chemical Engineering Ecole Polytechnique Federale de Lausanne, 2015

Submitted to the MIT Institute for Data, Systems, and Society
in partial fulfillment of the requirements for the degree of

Master of Science in Technology and Policy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
MIT Institute for Data, Systems, and Society
August 6, 2021

Certified by .....
Joann de Zegher
Assistant Professor, MIT Sloan School of Management
Thesis Supervisor

Certified by .....
Frank F. Field, III
Senior Research Engineer, MIT Institute for Data, Systems, and Society
Thesis Supervisor

Accepted by .....
Noelle Eckley Selin
Professor, Institute for Data, Systems, and Society and Department of Earth, Atmospheric and Planetary Sciences
Director, Technology and Policy Program

The Causal Impact of Information Crowd-sourcing Platform on Agricultural Supply Chain

by

Teuku Mahfuzh Aufar Kari

B.Sc. in Chemistry Nanyang Technological University, 2011

M.Sc. in Chemical Engineering Ecole Polytechnique Federale de Lausanne, 2015

Submitted to the MIT Institute for Data, Systems, and Society
on August 6, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Technology and Policy

Abstract

This thesis aims to study the impact of increased access to market information on the welfare of micro-firms in informal supply chains, as measured by revenue and price realization. In most informal supply chains, micro-firms receive access to only limited market information through their informal networks (e.g. SMS or Whatsapp). Previous literature finds that the introduction of mobile phones significantly improves market performance for perishable commodities but not so for non-perishable goods. We study whether tools that increase access to market information beyond informal information channels, e.g. through distributed price promotions or through crowd-sourcing, can further improve market performance. By leveraging transactions data of an information-sharing platform used by micro-firms in informal supply chains in a developing country and data from a pre-intervention survey, we seek to investigate the causal impact of the platform on the revenue of the users.

We found that the app leads to a significant increase the price (0.43%), post grading price (0.4%), and revenue (0.5%) realization of the sellers. We observed significant heterogeneity in post grading price treatment effect and new buyer rate. The heterogeneity in the former is driven by seller's learning, while the heterogeneity in the latter is driven

by local information availability.

Difference in means analysis revealed that the difference in post grading price treatment effect between users with high and users with low experience is around 80% of the average treatment effect. Linear regression indicates that as the user builds familiarity in using the platform, the conditional average treatment effect can grow by up to 7 times (from 0.2% to 1.6%). We found no evidence that information availability or prior commitment to sell to certain buyers affect the post grading price treatment effect.

Difference in means analysis indicated that heterogeneity in new buyer rate is driven by two factors, local information availability and prior commitment of the seller to sell to certain buyers. We found that the probability of transacting with a new buyer increases with local information availability but decreases with the presence of prior commitment. Regression analysis found statistically non-zero positive effect for local information availability, but not for prior commitment.

The observation that local information availability increases new buyer rate but not post grading price treatment effect can be explained by the fact that user may opt to switch to a new buyer for reason other than better pricing, e.g. shorter travelling distance. Due to difficulty in estimating transportation cost and travelling distance, our analysis in this work is limited to top-line impact only. Our analysis does not account for users who switched to a new buyer for travelling reason and benefit from the saving in transportation cost. Consequently, the results reported here are underestimation of the actual impact of the intervention.

Overall, we presented proof of concept for information crowd-sourcing platform as a low cost option to improve access to market information.

Thesis Supervisor: Joann de Zegher
Title: Assistant Professor, MIT Sloan School of Management

Thesis Supervisor: Frank F. Field, III
Title: Senior Research Engineer, MIT Institute for Data, Systems, and Society

Acknowledgments

"Nothing of me is original. I am the combined effort of everyone I've ever known."

Chuck Palahniuk, Invisible Monsters

One of my former colleagues told me that he thought it is very cool that I would go to MIT, that I would meet many "Tony Stark". To my own surprise, the "Tony Stark" I met at MIT is not the "Tony Stark" from the first Iron Man movie, who built Iron Man suit out of thin air in an Afghan cave. The "Tony Stark" I met is the "Tony Stark" from the last Avenger movie. They are the people whose compassion and kindness impressed me more than their brain power. I would like to thank the following people for the their kindness during my time at MIT:

- Joann, for hiring me to be her research assistant when I had very limited formal training in econometrics and for being very kind with me for the last 2 years.
- Frank, for being very generous with his time in reviewing my thesis and provided me with the English writing lesson that I wish I had before.
- The people who keep the TPP wheels running: Ed, Barb, and Noelle.
- All my fellow TPPers and MIT classmates who made MIT problem sets less daunting, Noelle's class less dry (IDS.412 Slack channel is unexpectedly amusing), and Frank's class surprisingly insightful.

I would have never been able to make it this far, if not for the help and constant support I received from friends and family. I would like to thank the following people for their numerous helps:

- The people who helped me get to TPP : Prof. Phan Anh Tuan, Cipto Herlianto, Brittany Benson, Prof. Djulia Onggo, and Brahim Heddi.
- My parents, Teuku Mohammad Alizar Kari and Cut Kuratul Aini Muli, for the unconditional love and for always giving their best in supporting all of my endeavours.
- All my siblings and their spouses who always love me despite my many shortcomings : Kak Ina & Bang Mahfud, Kak Ipit & Bang Ramli, Bang Is & Kak Yanti, and Kak Yudi & Bang Herman.
- The Indonesian community in Boston who made the pandemic and New England winter more bearable : Novia and Truong's family, Yodha and Dini's family, Yogi and Tia's family, Kifly and Ais's family.
- Sasha and Rasha who kindly let us stay in their place during summer 2021
- My in-laws who let me join the family and treat me like one of their own : Ayah Humam Hamid, Mamak Fitria Marzuki, Raisa & Alin, Razif & Riris, and Rais.
- All my nieces and nephews who survived and will probably continue to suffer from similar bad uncle jokes: Muhammad Zayyan Muhadzib, Muhammad Zharif Nafi', Muhammad Zakhir Makarim, Fatih Al Abiyyu Nasution, Hanifah Syakiroh Nasution, Cut Nazhara Ramadhina Kari, Cut Nazlia Rubiannina Kari, Zhafiratur Rahmah Felsia, Ihsanul Tsabit Felani.

It is easy to lose perspective at a place like MIT, and I would like to thank those who help me keeping my perspective when times are tough: my wife, Fairuziana Humam, and our "little sunshine".

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The best way to have a good idea is to have lots of ideas."

Linus Pauling

One of the key topics in international development is how to improve the productivity of smallholder farmers. Answering this question is of interest both in order to improve global food security and to promote sustainable and equitable growth that benefits smallholder farmers in developing nations. In this thesis, we explore this topic by pursuing a research question on whether the introduction of an information crowdsourcing platform that improves market access also leads to an improvement in the welfare of the platform's users. This document presents the results of our analysis.

This document is structured in the following manner. In Chapter 2, we will discuss the current state of the literature relevant to this work. The chapter will discuss the current consensus and open questions relevant to the impact of improvement in access to market information on the welfare of farmer. It will also discuss the recent development on the topic of sourcing ground truth in informal value chain. The chapter will

conclude with the formulation of the research questions outlined at the end of Chapter 2.

In Chapter 3, we will discuss the methodology used in pursuing the research question outlined in Chapter 2. We will start by explaining the empirical setup of this study, explaining the condition in the field and the data and information that we have access to. Afterward, we will discuss how we will identify and quantify the causal impact that we are interested in given the data constraints that we have. We will discuss in detail the data processing mechanisms as well as the statistical and computational methods used to pursue the research questions.

In Chapter 4, we will discuss the main results of this thesis. We will present evidence supporting our hypothesis that the use of the crowd-sourcing app increases the welfare of the farmers and agricultural intermediaries. A discussion on how and why the impact varies across users will follow.

Finally in Chapter 5, we will present the conclusion and main take-aways of this work, and how it contributes to the relevant literature.

Chapter 2

Literature Review

"The ink of the scholar is more sacred than the blood of the martyr."

Muhammad ibn Abd Allāh

In this section, we will outline the current literature on how information technologies have improved access to market information and the impact that they have brought to the smallholders farmers. We will then explain how this thesis work contributes to expanding the current literature on this topic.

2.1 Informal Agricultural Market

One of the key challenges in the 21st century is how to increase agricultural productivity to feed the growing world population. More than half of global calorie consumption, and up to 70 % of calorie consumption in developing nations, are supplied by informal smallholder households in the developing and emerging economies [26]. An informal smallholder farmer is defined here as a farmer who operates on a small-scale agricul-

tural model, in contrast to a large-scale corporate agricultural model. The importance of improving the productivity of informal smallholder farming has been recognized as key to meeting the global food demand [16].

Of the agricultural commodities that rely on smallholder farming, palm oil is among the fastest growing commodities in the world, with presence in more than half of consumer packaged goods [45]. The majority of global palm oil supply is produced by smallholder farmers. Indonesia produces 53% of globally-traded palm oil [44], with smallholder farmers being responsible for more than half of palm fruit production [32].

The rapid growth in demand for palm oil has also triggered a separate environmental concern, as more lands are converted to palm fruit production. In Indonesia, the major driver for carbon emission is land use, land use change, and forestry (LULUCF), causing the palm oil industry to face serious public pressure to improve their sustainability [18].

A typical schematic depicting the value chain for palm oil is given in Figure 2-1 on page 20. There are several distinct players in the value chain, which are listed below:

- The first player in the palm oil is value chain is the farmers, referring to people who cultivate the land to grow the palm fruits. Once the fruit is harvested, the farmer can choose to sell the fruit to transportation middlemen, to fruit aggregators, or directly to mill representatives.

- The second player in the value chain are the transportation middlemen, referring to people who buy the fruits from farmers and sell them to fruit aggregators or mill representatives.

- The third player in the value chain are the fruit aggregators, referring to people who buy fruit from farmer or transportation middlemen and transport them in a larger batch to the be sold to the mill representatives.

- The fourth player in the value chain are the mill representatives, referring to people who handle cash transactions on behalf of the mill. A single mill can be represented by multiple mill representatives and each mill representative may offer different prices to the sellers.

Farmers deciding where to sell their goods face a trade-off between maximizing revenue and minimizing cost. Revenue is driven by the price and, in some markets, the grading correction rate offered by the buyers. When farmers or other sellers sell the fruit to the buyers, their produce will be subjected to grading correction. Grading correction is a process where the buyer reduces the assessed weight of the fruit to take into account the quality of fruit. For example, if the grading correction rate is 3 %, that means for every 100 kg of fruit sold, the seller will only receive payment for 97 kg of the fruit. To maximize revenue, sellers seek to sell the fruit to buyers offering the highest price and the lowest grading correction rate.

One of the cost components for all the sellers along the value chain is transportation cost. Farmers who can transport the fruit by themselves may opt to sell directly to the mill representatives or the fruit aggregators, while the farmers without the means to transport the fruit by themselves may opt to sell to the transportation middlemen instead. The choice of whether to sell directly to the mill representatives or to the fruit aggregators is partly driven by the transportation distance.

Beside transportation cost, the decision on where to sell can also be affected by the credit relationship between different players. A farmer can borrow money from a transportation middleman, and in return he would pay off his loan by selling his fruit to the middleman at a lower price. As palm fruit can be seen as a commodity with little differentiation among farms, improving the process that goes into the decision on where to sell is key.

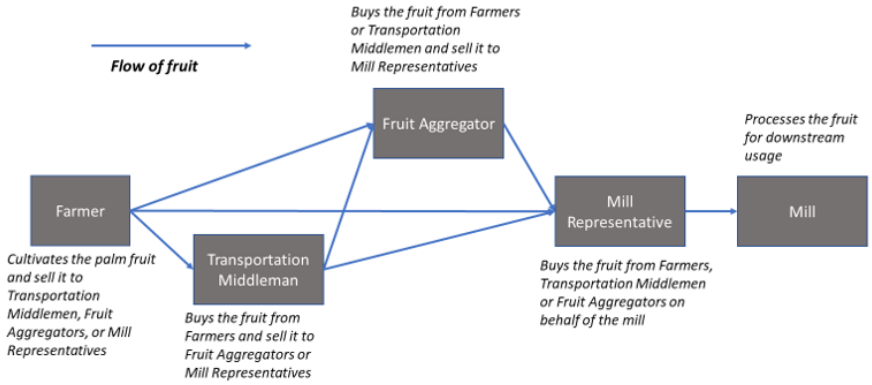


Figure 2-1: The schematic of palm oil supply chain

Another common feature of the informal palm fruit value chain is that a single person can wear multiple hats. For example, a farmer can buy fruits from other farmer and sell it to a mill, essentially acting as a transportation middleman as well as a farmer.

The key defining features of the informal palm fruit value chain is the lack of formal contracts and a low technology adoption rate. These characteristics have caused severe inefficiencies across the three pillars of efficient supply chains: flow of information, flow of goods and services, and flow of funds. Low technology adoption has hampered the flow of information across the value chain, resulting in information asymmetry and sub-optimal operation [35]. Consequently, the flow of physical goods is rendered sub-optimal by the lack of timely information. For example, a seller may be better off selling to a certain buyer, but he may be prevented from doing so because he has no price information about the buyer. Lack of formal contracts implies lack of proper documentation, which limits the access of value chain players to financial products. Given the importance of the informal agricultural value chain, the urgency of studying and im-

proving the functioning of informal market is clear.

2.2 The Impact of Improving Access to Agricultural Information

In comparison with other technological advances, mobile phone adoption has shown the fastest pace of adoption by the developing world [17]. It is estimated that one in three people have access to internet, and three of four people in low-and-middle-income countries (LMICs) own a mobile phone [17]. The adoption of mobile technologies is expected to continue rising as the cost of mobile phones decreases [9]. This trend presents an opportunity to leverage the use of information technology to improve agricultural productivity. One way that information technology can raise agricultural productivity is by enabling farmers to gain access to better agricultural practice.

To improve access to agricultural information, many government in LMICs employ extension workers to provide training and disseminate relevant information for the farmers. Some studies have shown that improving access to relevant agricultural information can contribute to increased agricultural productivity and lower environmental footprint. A randomized controlled trials (RCT) in China provided evidence that promoting integrated soil–crop system management among rural farmers using extension services can lead to improvement in crop yield and a reduction in greenhouse gas (GHG) emissions [46]. However, a World Bank study on the impact of agricultural extension service in Kenya during the 1982-1998 period found limited evidence of the cost effectiveness and the efficacy of the agricultural extension service, as it heavily relies on on-field human personnel [29]. Another study shows that extension agents tend to favor their own social network in providing services [31].

Another study in Mozambique shows that receiving extension services increases farm incomes by 12 %. However, the study also found that the extension services agents tend to target wealthier households that are relatively more likely to adopt the new technologies [5]. These sets of empirical findings have raised concerns that agricultural extension programs can lead to an increase in inequality. Other studies show the presence of gender bias in the delivery of extension services, that the services tend to benefit male farmers compared to female farmers [15] [24]. These bodies of empirical work suggest that, while extension services improve the agricultural knowledge of the farmers, the benefit of human-driven programs can be limited by cultural and human factors.

The rise of mobile phone adoption among smallholder farmers presents a new opportunity to improve access to agricultural information in a more cost effective manner, compared to a typical agricultural extension services. Furthermore, mobile phone based interventions may allow policy makers to remove the influence of the human agents and their associated biases. Evidence from several empirical studies have shown the impact of the mobile phone on improving the efficiency of agricultural extension services and promoting the adoption of better agricultural practices. Mobile phones have been shown to improve the impact of agricultural extension service, by improving the quality and the speed of delivery [47].

An RCT study based in Ecuador showed that using text messages can improve the adoption of integrated pest management by the farmers [7]. An RCT study based in India found evidence that an ICT advisory service leads to change in farming practice [38]. A meta analysis of six RCT suggests that SMS-based agricultural extension programs leads to a statistically significant increase in knowledge, albeit the study found no evidence that increases in technological adoption is sustained over the following seasons [35].

The adoption of better agricultural practice has also been shown to improve agri-

cultural yield. A study found that sending SMS messages with agricultural advice to smallholder farmers increased yields by 11.5% [25].

Apart from promoting better agricultural practice among farmers, improvement in access to information can also lead to market level impacts, such as allowing a farmer to sell their produce to seller at a higher price and reducing price dispersion in the market. In accordance to the Law of One Price, improved access to information has been shown to lead to the reduction in price dispersion of fish in India, of grains in Niger, and of crops in India [36],[19],[20],[8].

Improvement in access to market information may also improve the bargaining power of the farmers, allowing them to negotiate better price or sell their produce to buyers with better price. A study in Uganda found evidence that the Management Information Service (MIS) project led to improved bargaining power, which leads to higher farm-gate prices [22]. A field experiment in Peru estimated that farmers who received price information received 13-14 % higher prices for their produce [13]. A study on the causal impact of MIS in Ghana shows that the MIS program significantly increased price for maize (10 %) and groundnuts (7 %) [33]. A study in India found that the introduction of an internet kiosk led to a significant increase in soy price [4].

However, some studies have found that improved access to information has limited effect on the price received by the farmers. A case study in Rwanda found that improved access to telephone did not lead to a significant change in price [28]. The study on the impact of Reuters Market Light (RML) found no statistically significant impact on the price received by farmers, crop value-added, crop losses resulting from rainstorms, or the likelihood of changing crop varieties and cultivation practices [27]. Another study in Colombia on the impact of SMS based dissemination of price information also found no statistically significant impact in the actual sale price, farmer's revenues or household expenditure [3]. A study in West Bengal, India, found that improved access to price

information had no significant impact on average farm-gate sales and prices, but it increased pass-through to farm-gate prices [40].

The fact that empirical results on the impact of improved access to price information on the price realization are mixed is due to a combination of differences of target population, the nature of the crops, message design, informational constraints, and the barriers to effective use of ICT [35]. The first condition for an improvement in access to information to have an impact is the presence of market inefficiency. In a perfectly efficient market with no price dispersion, improved access to information is not expected to improve price realization of the farmers. The study on the impact of telephony roll-out in Rwanda attributed the lack of impact on the farmer welfare to the fact that most of the villages that received telephony roll-out were already sufficiently efficient, e.g. the beneficiaries are already entrepreneurial, hence the opportunity for improvement is small [28]. Another condition is that the farmers need to have the capacity to act on the new information that they receive. In some markets, farmers lack the means of transportation and relationship to transport and sell their produce to sellers offering higher prices [19],[20],[27],[40].

2.3 Sourcing Ground Truth Information in Informal Market

Equally important to the question of how to improve market information for the farmer is the question of how to source the market information in the first place. Most agricultural extension services and MIS tools discussed in the previous section rely on centralized information collection, which can be provided by the government or private entities (e.g. Reuters Market Light). In a typical informal market, market information is

scattered across market players and price information changes on a daily basis. In such cases, manual collection of information is expensive, and in some cases, impossible to collect in real-time [8]. It is estimated that the ratio of rural farmers to extension workers exceeds 1000 in many parts of the globe, highlighting the dire need to find a scalable means of data collection [43]. Even in the presence of sufficient budget to pursue manual collection of information, market players may be reluctant to share information in the absence of proper incentives [12].

One way to collect market information that has been shown to be cost effective and scalable is via information crowd-sourcing mechanism [11]. Information crowd-sourcing is defined as a mechanism which collects information from a network of contributors that are not doing so as part of their normal professional activities. The information contributors may or may not receive any financial rewards for their contribution [14].

In agriculture, there have been several crowd-sourcing projects that aim to promote better agricultural practice among the farmers, often in the form of an internet forum. Croprotech aims to provide UK farmers with information related to weeds, pests and diseases [42]. FarmHack, (www.farmhack.org), is a community platform that allows users to discuss their agricultural machine needs. The Pommak initiative by the Belgian extension service aims to gather and disseminate price information among farmers (www.pommak.be). In the context of rural smallholder farmers in developing nations, KON is an example of the use of crowd-sourcing mechanism to gather environmental information [23].

2.4 Research Questions

We identified two literature gaps that this work seeks to address. Firstly, the crowd-sourcing mechanism has gained attention as a scalable and cost-effective way to improve access to agricultural information. However, the literature does not assess the impact of the crowd-sourcing mechanism on the welfare of market players. Hence, the first research question that we pursue in this work is : **what is the impact of improving access to information via information crowd-sourcing platform on the welfare of the users?**

Secondly, there appears to be a conundrum on whether improving access to relevant market information will leads to improvement in the welfare of market participants. While most empirical work seems to report that improvement in access to relevant market information will lead to a reduction in price dispersion, the impact on the welfare of the market player is heavily context dependent. Studying which context will lead to welfare improvement is of particular interest to policy makers.

Most of the the empirical works in this topic are limited by two factors. Firstly, most of these studies used survey data with limited temporal granularity and may be compromised by reporting and memory bias. Secondly, the unit of analysis in most of these studies is set at low granularity, i.e. the object of interest or the unit of analysis is the price at a certain location.

In this work we used the data from an anonymous information crowd-sourcing platform that allows farmers to share transaction information with one another. The platform was only recently launched in December 2018, and we used the data from the launch until February 2021. The detailed description of the platform partner is given in the Section 3.1 of the next chapter. This data has finer temporal granularity than a typical survey data and allow us to set the unit of analysis at a finer level, i.e. our unit

of analysis is the price received by a person. Leveraging this data, we aim to pursue the second research question, which is : **what are the factors that leads to improvement in welfare?**

Chapter 3

Methodology

"Science may be described as the art of systematic oversimplification — the art of discerning what we may with advantage omit."

Karl Popper

This section describes the empirical setup, the methods used for data preparation, and the statistical methods used to perform data analysis.

3.1 Empirical Setup

Our partner is a mobile-based supply chain management technology company for informal supply chains, whose products include an information crowd-sourcing mobile app platform for smallholder palm oil farmers and middlemen in a developing country. Due to the agreement that we have with the partner, the identity of the company and the app will not be disclosed. The platform enables users to view and share transaction information including price, grading correction rate, transaction experience, and

availability of cash. In this work, we mainly used the data from the back-end server which records the usage activities of the farmers and middlemen, such as pageviews and information sharing activities.

There are four classes of users in the platform, which are characterized based on the role that they play in the supply chain, as listed below:

1. **Farmer**. This class refers to the users who cultivate the land and produce palm fruit. After harvesting the fruit, a farmer has the option to either transport the produce to the buyer's location by themselves or to sell it to a transportation middleman.

2. **Transportation Middleman**. This class refers to the users who act as intermediaries between farmers and downstream buyers. A transportation middleman buys the fruit from farmer and transports them to either a ramp or a mill.

3. **Ramp Manager**. This class refers to the users who manage a ramp and act as intermediaries who buy the produces from farmers and transportation middlemen, and then transport and sell them to a mill. A key differentiation between a ramp manager and other transportation intermediaries is the scale of operation. A ramp manager aggregates the fruit from farmers and middlemen to a high enough volume that makes transportation to a more distant mill worthwhile. Ramp manager uses the app to gain access to the prices at different mills and uses the information to decide which mills or which mill representatives to sell to.

4. **DO (Delivery Order) Representative of the Mill**. This class refers the mill representatives, called DO representative, who handle cash transactions on behalf of the mill. A DO representative buys the produce from farmers, transportation middlemen, and ramp managers, on behalf of the mill that he represents. The DO

representatives that belong to the same mill may set slightly different prices.

The pictorial representation of the relationship among this different classes is presented in Figure 2-1 on page 20.

Our platform partner offers several features to the users, the most salient one being the information crowd-sourcing feature. This feature allows the users to share and look for price information. Buyers (ramp manager and DO representative) can use the app to broadcast the price that they are willing to offer to the sellers. Sellers (farmer, transportation middleman and ramp manager) can use the app to see what prices are being offered for palm fruit so they can decide where to sell. The sellers can also use the app to share the price information that they receive from buyers, which is the essence of the crowd-sourcing feature.

A particular feature of the informal palm fruit value chain is that several sellers and buyers can form a closed group known as a "special price group". A special price group consists of a buyer and several sellers. Sellers that belong to this group will receive a special price from the buyer, and this price typically differs from the price that the buyer offers to other users. The buyers can use the app to form one of these special price groups and broadcast the special price to the corresponding members. This so-called "special price" can be either higher or lower than the typical price depending on the circumstances. For example, a seller can receive special price because he is a loyal seller, because he tends to deliver higher-quality fruit, because he tends to deliver higher volumes of fruit, or because he has an informal loan to pay off.

From our platform partner, we gained access to three types of data, which are listed below:

1. **The transaction receipt data**. The transaction receipt data includes, for each sale, the following information:

- the identity of the buyer,
- the identity of the seller,
- the sales price of the transaction,
- the tonnage of fruit sold, and
- any grading correction rate applied.

Using this information, we can estimate the nominal price, the post grading price, and the revenue that the users receive.

2. **The in-app user activity data**. The in-app user activity data includes two types of information:

   - the mobile app page view activity, and
   - the information sharing activity of the users

   Using this information, we can study users behaviors and understand which prices the users viewed (and therefore took into consideration).

3. **The user demographic data**. The user demographic data is the user specific information. This information includes:

   - the list of buyers from whom the user received price information prior to using the app

   Using this information, we can estimate the price, post grading price, and revenue that the user would have received if the user had not joined the app.

3.2 Causal Framework

3.2.1 Potential Outcome Formulation

In an ideal setting, our research question on the causal impact of information crowdsourcing platform on the welfare of the users could be answered by comparing the welfare outcome that the users attained after using the app (factual) with the welfare outcome that the users would have attained if they hadn't used the app (counterfactual). As an illustration, for a user who attained revenue of 100,000 IDR after using app and would have attained revenue of 80,000 IDR if he had not used the app, the treatment effect is calculated as 100,000 IDR (factual outcome) - 80,000 IDR (counterfactual outcome) = 20,000 IDR. Mathematically, this approach is expressed using the potential outcome framework of Rubin (1974), which is presented in Equation 3.1 [6]. In this study, the **treatment** is defined as the use of the app by the user, and the **treatment effect** is defined as the difference between the observed outcome and the outcome that we estimate would have occurred in the absence of the mobile app (the treatment).

$$\text{TreatmentEffect} = \text{FactualOutcome} - \text{CounterfactualOutcome} \tag{3.1}$$

In this study, we consider several outcome variables in our analysis, which are listed in Table 3.1 on page 36. Out of those variables, we are mainly interested in three welfare outcome variables: price, post grading price, and revenue. To estimate the treatment effect, we need to estimate both the factual outcome and counterfactual outcome. In this study, the factual outcome measurement is gleaned from the reported transaction receipt data on the platform.

One perennial problem in causal effect quantification is that only the factual outcome is observed. In a typical randomized control trial setting, there is a comparable

control group from which the counterfactual outcome can be estimated. However, in this study, we only have access to transactions records of the users of the mobile app, and there is no control group.

Given this limitation on data availability, we need to use two identification assumptions to estimate the counterfactual outcome. Firstly, we assume that, in the absence of treatment, the user will only have access to the market information of his counterfactual buyers set, i.e. the set of buyers from whom the users received price information prior to joining the platform. Consequently, the counterfactual outcome of the user can be expressed as a function of the price of his counterfactual buyers set (e.g. mean, max). Secondly, we assume that treatment does not affect the price set by the buyers. As a direct consequence of this assumption, we can assume that the price set by the buyers in the absence of treatment is the same as the price that we observed.

In estimating the treatment effect, we use three different scenarios: the average case, the conservative case, and the special group robust case. The differences in each of the three cases are presented below:

- **Average Case**. This case serves as the main analysis case. The counterfactual values are calculated as the average of values of the variable (e.g., price, post grading price, etc.) for all of the subset of counterfactual buyers on the day of the transaction.

- **Conservative Case**. In this case, the counterfactual values for price, post grading price, and revenue are calculated as the maximum of available values of all the counterfactual buyers on the day of the transaction. By taking the maximum instead of the mean, we can test whether the estimated treatment effect is still significant if we assume that the user receives the best possible counterfactual outcome, instead of the average of all of his possible counterfactual outcomes.

- **Special Group Robust Case**. In this case, the counterfactual values for price, post grading price, and revenues are calculated in similar manner as the mean case. The only exception is that the set of counterfactual buyers includes all the special group buyers that the users belong to. The motivation for running this scenario is an effort to mitigate the effect of unreported relationships between users and buyers. Based on field experience, if a user belongs to a special group with a buyer, then it is very likely that user and the buyer had a relationship prior to joining the platform. However, there are some cases where a user-buyer pair belongs to the same special price group, but the buyer is not reported as part of the user's counterfactual buyer set. Hence, we want to test whether the estimated treatment effect is still significant if we assume that a buyer who belongs to the same special price group as the user is considered to be part of his counterfactual price buyers.

Table 3.1: Variable description

Variable	Realized Definition	Counterfactual Mean Case	Counterfactual Max Case	Counterfactual Special Group Robust Case
Price	The price realized by the user	The average price that the user would have access to on the day of transaction that comes from his old circle of buyers	The highest price that the user would have access to on the day of transaction that comes from his old circle of buyers	The same as counterfactual mean case, except the the set of counterfactual buyers includes all buyers in the user's special group memberships
Best Price	The highest price that the users could have realized based on the price available in the platform on the day of transaction	The highest price that the user would have access to on the day of transaction that comes from his old circle of buyers		
Best Price Viewed	The highest price that the users viewed in the platform on the day of transaction	The highest price that the user would have access to on the day of transaction that comes from his old circle of buyers		
Post Grading Price	The price realized by the user, after weight grading correction	The average price, after weight grading correction, that the user would have access to on the day of transaction that comes from his old circle of buyers	The highest price, after weight grading correction, that the user would have access to on the day of transaction that comes from his old circle of buyers	The same as counterfactual mean case, except the set of counterfactual buyers includes all buyers in the user's special group memberships
Best Post Grading Price	The highest price that the users could have realized based on the price available in the platform on the day of transaction, after weight grading correction	The highest price, after weight grading correction, that the user would have access to on the day of transaction that comes from his old circle of buyers		
Best Post Grading Price Viewed	The highest price that the users viewed in the platform on the day of transaction, after weight grading correction	The highest price, after weight grading correction, that the user would have access to on the day of transaction that comes from his old circle of buyers		
Grade	The weight grading correction rate that the user received	The average weight grading correction rate that the user would have received on the day of transaction that comes from his old circle		

Variable	Realized Definition	Counterfactual Mean Case	Counterfactual Max Case	Counterfactual Special Group Robust Case
Revenue	The revenue that the user received from the transaction	The average revenue that the user would have received, based on the average price of the sellers in his old circle of buyers	The revenue that the user would have received if he transacted with the buyer who offered the highest counterfactual post grading price	The same as counterfactual mean case, except the the set of counterfactual buyers includes all buyers in the user's special group memberships

3.3 Data Processing

This section describes how each of the tables used in this work are constructed from the original database tables of our platform partner. The raw tables that we received from our platform partner will be described in Section 3.3.1. The intermediaries and the constructed tables will be discussed in the subsequent sections.

3.3.1 Source Table Description

In this work we used (1) several tables from the our platform partner database, (2) a survey administered in 2018, and (3) a table containing manually gathered information. The list of tables that we used and their descriptions are listed below.

1. Tables from the app platform partner:

    - ***receipt***. This table contains information on receipt (record of sales transaction) uploaded by the users. Each row represents a receipt uploaded by the user.
    - ***transactions***. This table contains sales transaction information contributed by the users. The user can share price, grading correction rate, weight, and cash availability for a given buyer. Each row represents either a price, grading correction rate, weight, or cash availability by the user.
    - ***fruit_sale***. This table contains sales transaction information contributed by the users. The user can share price, grading correction rate, weight, and cash availability for a given buyer. This a newer version of *transactions* table. Each row represents a transaction information share by the user containing all the above mentioned information (if available).

- ***last_most_transaction_do_ramp***. This table contains the information on prior relationship between buyers and sellers. When a new buyer joins the platform, he will be asked to complete a questionnaire listing all the buyers from whom he received price information prior to joining the platform. If a seller receives price information from a particular buyer prior to joining the platform, then the buyer and the seller will be documented in this table. Each row represents a user and the buyer from whom he received price information prior to joining the platform.
- ***price_group_price***. This table contains information on the price given to the users from the associated special price group. Each row represents a user and the price offered to them on a given date by a specific buyer.
- ***price_group_user***. This table contains information whether a user belongs to any specific special price group or not. Each row represents a user and his associated price group membership.
- ***do_login_activities***. This table contains information on browsing activity of the users on the DO representatives pages. Each row represents a pageview activity on a given DO representative page and contains information on the user viewing the page and the timestamp.
- ***ramp_login_activities***. This table contains information on browsing activity of the users on ramp pages. Each row represents a pageview activity on a given ramp page and contains information on the user viewing the page and the timestamp.

2. Table from the field survey:

    - ***user_old_buyers_survey***. This table contains information from a survey con-

ducted in 2018 which asked the seller whether they know a particular buyer or not. Each row represents a user and the mill that he knew at the time of the survey.

3. Table from the manually gathered information:

    - ***users_receipt_duplicates***. This table contains information on the manually gathered information from the receipt pictures. It contains information whether a particular receipt is a duplicate. It also contains price, grading correction rate, and weight information.

3.3.2 Construction of *users_receipt* Table

The *users_receipt* table contains every fruit sales transaction analyzed in this work. Each row of the table corresponds to a single transaction reported by the user. Each transaction is defined as an exchange of fruit for money accompanied by a physical receipt recording the transaction. The unique row identifier is given in *receipt_id* column. To guarantee the integrity of the data, we performed a manual removal of duplicates. The manual removal of duplicates is done by visually comparing the physical receipt uploaded by the users with one another. If two or more receipts are the same, then only one of them will be considered for the downstream analysis. Due to the structure of the database, we constructed the user receipt table by combining transaction data from three different tables as listed below:

- Data from the legacy *receipt* table, selecting only 1 row per actual sales transaction. If a given transaction has multiple rows, than only 1 row is considered.

- Data from *transactions* table that was not found in the legacy *receipt* table.

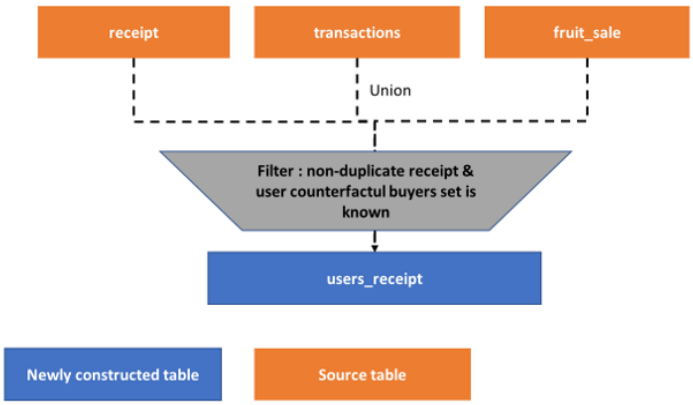


Figure 3-1: *users_receipt* table construction schematic

- Data from *fruit_sale* table.

In the *users_receipt* table, we only include receipts that we will use in the analysis. Hence, we filter the rows from the original source tables to include only the non-duplicate transaction receipts and transaction receipt of users whose counterfactual buyers set is known.

The schematic depicting the construction of *users_receipt* table is presented in Figure 3-1. The detailed method for constructing *users_receipt* table is presented in the *Notebook 5* in the appendices.

3.3.3 Construction of *counterfactual_price* and *counterfactual_price_buyer_receipt* Table

In estimating both the factual and counterfactual outcome, we need to have a table which organizes the daily price and grading correction rate of each buyer. Hence, we constructed the *counterfactual_price* table, which contains the daily market informa-

tion (price, grading correction rate, post grading price) of the buyers in the platform. To construct the daily price table, we based our data source on all the reported prices from all users, both the data shared directly from the buyers as well as the data that come from the sellers as they report their transactions. All the data used for this work is based on the user contributed information on our partner information crowd-sourcing platform from December 3 2018 to February 19 2021.

After consolidating all the market information contributed to the platform, we removed outlier data from the table. Outlier removal is a process of removing data point that differs significantly from other observations and, moreover, its removal is based on an assumption that this difference is the result of a data error. This process was done using a 2-pronged approach. Firstly, we defined cross sectional outliers as those price points that differ significantly from other data points on the same date. For this consideration, we removed all data points whose values differ by more than 400 IDR compared to the mean price in a given date. We picked 400 IDR as the threshold because there are less than 0.5% of the data points in this region, but it is present in the distribution as a long tail. Additionally, based on the experience of our platform partner, we believe a price difference of more than 400 IDR is unlikely, and hence we opt to exclude them. Secondly, we define time-series outliers as those price points that significantly differ from the previous date's or the next date's price points. For this type of outlier, we removed those price points whose values differ for more than 500 IDR compared to the neighbouring points. Similar to cross-sectional outliers, we picked 500 IDR as the threshold because there are less than 0.5% of the data points in this region, but it is present in the distribution as a long tail. We also believe that a price fluctuation of more than 500 IDR is unlikely, and hence we opt to exclude them. The histogram on the data distribution is presented in Figure 3-3. In total, we removed 47 price points from 95,918 points.

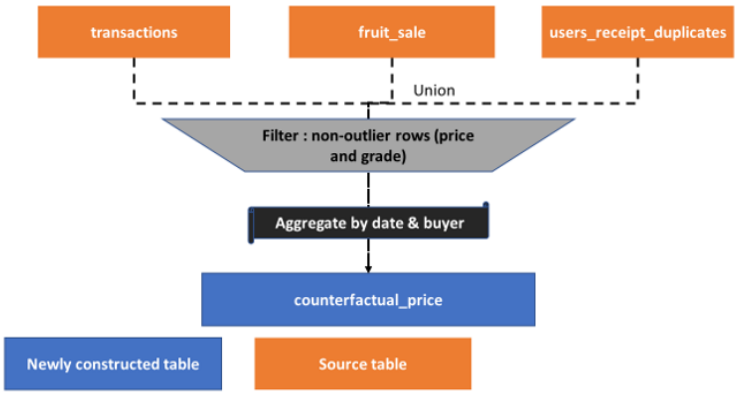


Figure 3-2: *counterfactual_price* table construction schematic

the daily price table for each buyer was constructed by taking the median of all prices reported for a given buyer on a given date.

As is typical in any empirical work, we found a lot of missing price and grading correction rate information in our dataset. For DO data, we found that the DO representatives that belong to the same mill tend to offer similar grading correction rate, as the grading correction rate is usually set by the mill. As such, we used the grading correction rate data from other DO with the same mill to impute the missing grading correction rate data. To do so, we need to construct *counterfactual_price_buyer_receipt* table, which contains daily price and grading correction rate information for each mill and ramp manager, but not DO. This table can be viewed as a less granular version of *counterfactual_price* table.

To construct the *counterfactual_price_buyer_receipt* table, we need to generate the unique *buyer* coding. To generate unique coding for each *buyer* in our analysis, we concatenate the type of transaction ('mill' for mill transaction and 'lr' for ramp transaction),

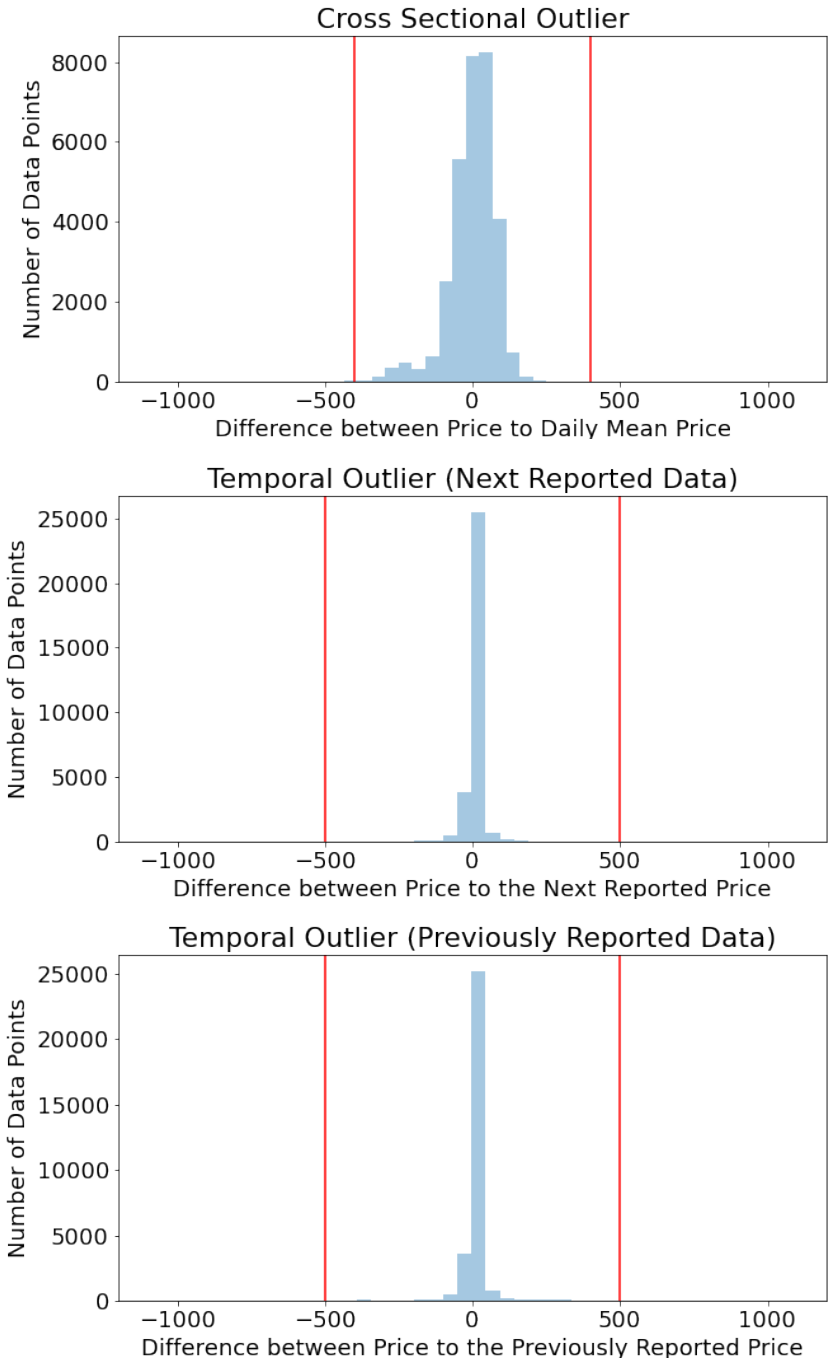


Figure 3-3: The histogram of price data
Vertical red lines indicate threshold for outlier removal

the *mill_id*, the *do_id*, and the the *ramp_id*, separated by "//". For example, the buyer coded as *"mill//1//3//nan"* corresponds to a DO with *do_id* 3 who is affiliated to *mill_id* 1. The *ramp_id* is "nan" because a DO does not have a *ramp_id*. The "nan" here is the abbreviation for "Not a Number". In the legacy *receipt* table, the DO information is not recorded. As such, for some of the transaction, the DO information is missing. We defined a separate variable called *buyer_receipt*, referring to to the buyer as reported in the physical receipt uploaded by the users. For a ramp transaction, the *buyer* coding and the *buyer_receipt* coding are the same. However, for the mill transaction, the *buyer_receipt* coding is the same as *buyer* coding, except the *do_id* is set to "nan".

The *counterfactual_price_buyer_receipt* is constructed in similar manner as *counterfactual_price* table. Except instead of *buyer*, it contains the market information for every *buyer_receipt* for every day.

The schematic depicting the construction of constructing *counterfactual_price* and *counterfactual_price_buyer_receipt* table is presented in Figure 3-2 on page 43 and Figure 3-4 on page 46, respectively. The detailed method in constructing *counterfactual_price* and *counterfactual_price _buyer_receipt* table is presented in the *Notebook 4* in the appendices.

3.3.4 Construction of *do_ramp_visit* Table

The *do_ramp_visit* table contains every DO representative and ramp page visited by the users while the users is browsing the app. The *do_ramp_visit* table is constructed by combining the information from the *do_login_activities* and *ramp_login_activities* table from the platform partner database.

The schematic depicting the construction of *do_ramp_visit* table is presented in Figure 3-5.The detailed method in constructing *do_ramp_visit* table is presented is the

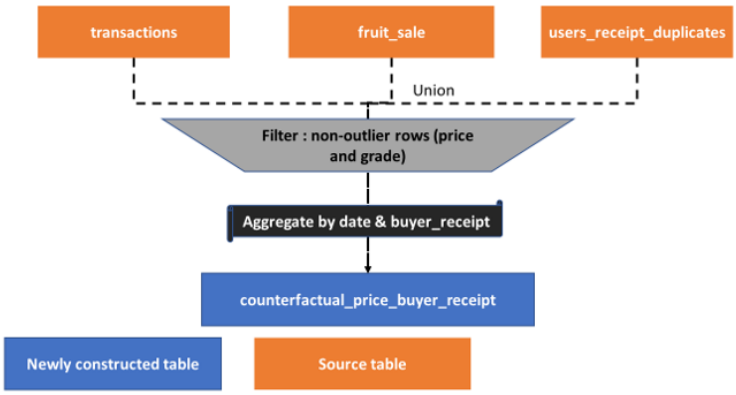


Figure 3-4: *counterfactual_price_buyer_receipt* table construction schematic

Notebook 1 and *Notebook 4* in the appendices.

3.3.5 Construction of *old_buyers_df* Table

To estimate the counterfactual outcome for each user, we defined a set of counterfactual buyers. we use three sources of data used to construct the counterfactual buyer set:

- The field survey administered in 2018. In this survey, the users were asked whether they knew a specific buyer or not. This survey was administered only to a small subset of the users, 17 users out of 1,181 users whose counterfactual buyers set are known.
- The in-app survey embedded in the platform. Since November 2019, all the new users who join the platform were asked to complete a survey where the users was asked to list up to 6 buyers whom they received market information prior to using the app.

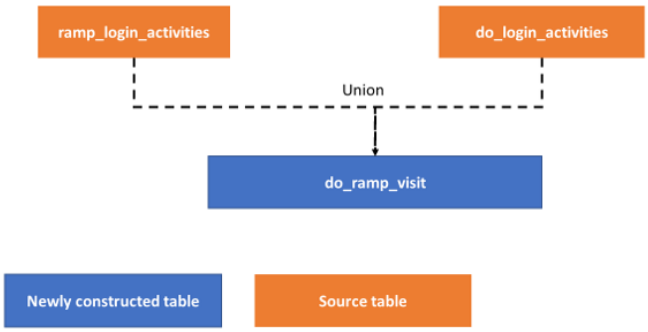


Figure 3-5: *do_ramp_visit* table construction schematic

- The buyers with special group relationship to the users as recorded in the special group relationship table. The platform has a feature that allows the buyers to offer a special price to a select group of sellers.

We combined the old buyers information from the three data sources to form a table containing every relationship between each user and their old buyers.

The schematic depicting the construction of *old_buyers_df* table is presented in Figure 3-6. The detailed method in constructing *old_buyers_df* table is presented in the *Notebook 5* in the appendices.

3.3.6 Construction of *special_price_daily* Table

The *special_price_daily* table contains the special price information offered from the buyers to the sellers. The special *special_price_daily* table is constructed by outer joining the *price_group_price* and *price_group_user* table using the user_id of the seller and the user_id of buyer as the key.

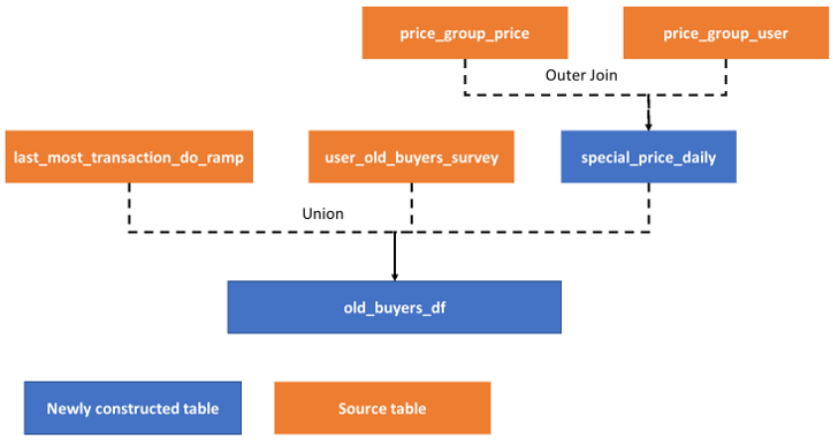


Figure 3-6: *old_buyers_df* and *special_price_daily* table construction schematic

The schematic depicting the construction of *special_price_daily* table is presented in Figure 3-6. The detailed method in constructing *special_price_daily* table is presented in the *Notebook 4* in the appendices.

3.4 Variable Calculation

The *users_receipt* table contains only the transaction used in the analysis, with each row representing a unique transaction from the users whose counterfactual buyers set are known. To perform the analysis, we need to calculate the relevant variables for each row in the *users_receipt* table. This section describes the method used for the calculation of outcome and counterfactual variables and the imputation of any missing data.

3.4.1 Price and Post Grading Price Calculation

The price and grading correction rate information for each transaction in *users_receipt* table is obtained from the values reported by the users. Different users transacting with the same buyer may report different prices because the buyer may quote different prices for the users depending on several factors, such as the quality of the fruit or whether the sellers has outstanding debt to the buyer. In the case where the user has an outstanding debt to the buyer, the buyer may pay a significantly lower price per kg of the fruit, as the remaining price difference is used to pay off the debt.

As there are multiple sources of price information, the hierarchy of the price information is as follows. The first source of price is the manually imputed information that is sourced from the *users_receipt_duplicates* table. The second source of price information comes from the *special_price_daily* table which contains the price information given to the user from the special price group that they belong to. The third source of price information is the *fruit_sale* table. The fourth source of price information is the *transaction* table. The fifth source of price information is the *counterfactual_price* table. The hierarchy works by using the data information from the source with the highest hierarchy (first source), and using the subsequent source to impute the missing data.

The principle used to determine the hierarchy between the data sources is the granularity and trustworthiness of the source. In this case, the price data from the *users_receipt_duplicates* table has the finest granularity and the highest trustworthiness, because it was gathered by visually inspecting each physical receipt. On the other end, the data from the *counterfactual_price* table has the lowest granularity because it only contains the median price of the buyer on the day of transaction. Hence, using the price data from the *counterfactual_price* table loses some of nuance of the actual transaction. We only use price data from the *counterfactual_price* table in cases where all other data

Figure 3-7: The data hierarchy for price imputation

sources have been exhausted.

The graphical representation of the hierarchy is presented in Figure 3-7. For example, if we want to determine the price data for buyer "mill//76/18//nan" on February 19, 2020, we first look for this information on the *users_receipt_duplicates* table. If the data does not exist on the *users_receipt_duplicates* table, then we go to the next table on the hierarchy until we found the data. The illustrative table depicting an example the result of data imputation is given in Table 3.2. As can be seen from the table, a single transaction can have multiple price data reported from different original source tables and the newly constructed *users_receipt_duplicates* table. In some cases, the price data reported from different table may slightly differ, as shown in row 2 and row 4. In such cases, the imputation hierarchy is used to determine which data is to be used for analysis.

Table 3.2: Illustrative table for the result of data imputation

Row	Price from users_receipt _duplicates table	Price from special_price _daily table	Price from fruit_sale table	Price from transaction table	Price from counterfactual _price table	Price data used for analysis	Chosen data source for price
1	NaN	NaN	NaN	NaN	940	940	counterfactual _price table
2	905	NaN	NaN	NaN	903	905	users_receipt _duplicates table
3	NaN	NaN	NaN	1000	1100	1000	transaction table
4	NaN	NaN	1800	NaN	1785	1800	fruit_sale table
5	NaN	1970	NaN	1970	1970	1970	special_price _daily table

Figure 3-8: The data hierarchy for grade imputation

The hierarchy for grading correction rate information is almost similar to the hierarchy for price information. The first source of grading correction rate is the *users_receipt _duplicates* table. The second source of grading correction rate information is the *fruit_sale* table. The third source of grading correction rate information is the *counterfactual_price _buyer_receipt* table. Lastly, the *fruit_sale* table reports the adjusted weight and the total weight for each transaction. Using this information, we calculated the grading correction rate for the transaction, labelled as $Grade$, using Equation 3.2. The fourth source of grading correction rate information is the grading correction rate information obtained from this calculation. To ensure data integrity, we removed records with a grading correction rate less than 1, as we believe that this information may have been incorrectly entered. The graphical representation of the hierarchy is presented in Figure 3-8 on page 52.

$$\text{Grade (\%)} \equiv \frac{\text{AdjustedWeight (kg)}}{\text{TotalWeight (kg)}} \times 100\% \tag{3.2}$$

The hierarchy for post grading price information is as follows. The post grading price information for each transaction is calculated by Equation 3.3 using the price and grading correction rate information available for the transaction. The first source of post grading price information is the value calculated using this method. The second source of price information is the *counterfactual_price* table.The graphical representation of the hierarchy is presented in Figure 3-9 on page 53.

Figure 3-9: The data hierarchy for post grading price imputation

$$\text{PostGradingPrice (IDR/kg)} = \text{Price (IDR/kg)} * (1\text{-Grade (\%)}/100\%) \tag{3.3}$$

The detailed method for this calculation is presented in the *Notebook 5* in the appendices.

3.4.2 Best Price and Best Price Viewed Calculation

The best price information is calculated by taking the maximum reported price on a given date from the *receipt* table, *transaction* table, and *fruit_sale* table. The best price viewed information is calculated by taking the maximum price that a user views on a given date based on the user browsing activity documented in *do_ramp_visit* table and *users_receipt* table. Presumably due to some glitch in the database of our platform partner, some of the receipt upload behaviours are not reported in the *do_login_activities* or *ramp_login_activities* tables. To compensate for this, we infer that the best price known to a user in a given date is the maximum of the price seen by the user as documented in the *do_ramp_visit* table and the price reported by the user in the *users_receipt* table. The calculation for best post grading price and best post grading price viewed is done in a similar manner.

The detailed method for this calculation is presented in the *Notebook 5* in the appendices.

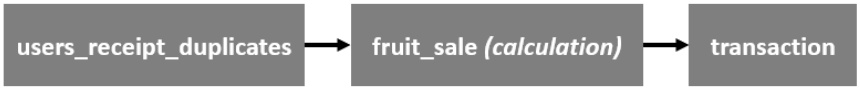


Figure 3-10: The data hierarchy for post grading weight imputation

3.4.3 Weight and Revenue Calculation

Due to the presence of grading correction described in Section 2.1, the weight of the fruit sold is reported in the platform in several terms: total weight, adjusted weight, and post grading weight. Total weight refers to the total weight of the fruit sold, adjusted weight refers to the weight deducted from the total weight as part of the grading correction, and post grading weight refers to the final weight of the fruit to which the farmers receive payment.

As there are multiple sources of weight information, we applied an information source hierarchy procedure similar to that used for price information. The first source of post grading weight information is the *users_receipt_duplicates* table. The second source of post grading weight information is the *fruit_sale* table, where the post grading weight information is calculated as the total weight minus adjusted weight, as presented in Equation. 3.4. The third source of post grading weight information is the *transaction* table. The graphical representation of the hierarchy is presented in Figure 3-10.

$$\text{PostGradingWeight (kg)} = \text{TotalWeight (kg)} - \text{AdjustedWeight (kg)} \tag{3.4}$$

The revenue for each transaction is calculated as the product of post grading weight and price, which is presented in Equation 3.5.

$$\text{Revenue (IDR)} = \text{Price (IDR/kg)} * \text{PostGradingWeight (kg)} \tag{3.5}$$

The detailed method for this calculation is presented in the *Notebook 5* in the appendices.

3.4.4 Counterfactual Price, Post Grading Price, and Revenue Calculation

There are two main steps in calculating the counterfactual variables: (1) determining whether the transaction is a new buyer transaction and (2) determining the values of the counterfactual variables. In determining whether the transaction is a new buyer transaction, we followed the procedure described in Figure 3-11 on page 56. The red boxes in the Figure 3-11 represent the cases where we are required a conservative assumption because either the mill, the DO, or the ramp is unknown. Out of the three possible cases for this, we only encountered one case : the case where the DO is unknown which amounts to around 15 % of all transaction analyzed.

The counterfactual variable definition for each scenario is presented in Table 3.1 on page 36. If a transaction is a not a new buyer transaction, then the corresponding counterfactual is the same as the observed variable, and the treatment effect is zero by definition. If the transaction is a new buyer transaction then the corresponding counterfactual variable is calculated as per the variable definition in Table 3.1. The source of old buyer information is the *old_buyers_df* table. The *old_buyers_df* table contains information on whether the old buyer is reported by the user from the in-app survey and the 2018 field survey or whether it comes from the special group membership. In the mean counterfactual and max counterfactual cases, we only used the the buyers re-

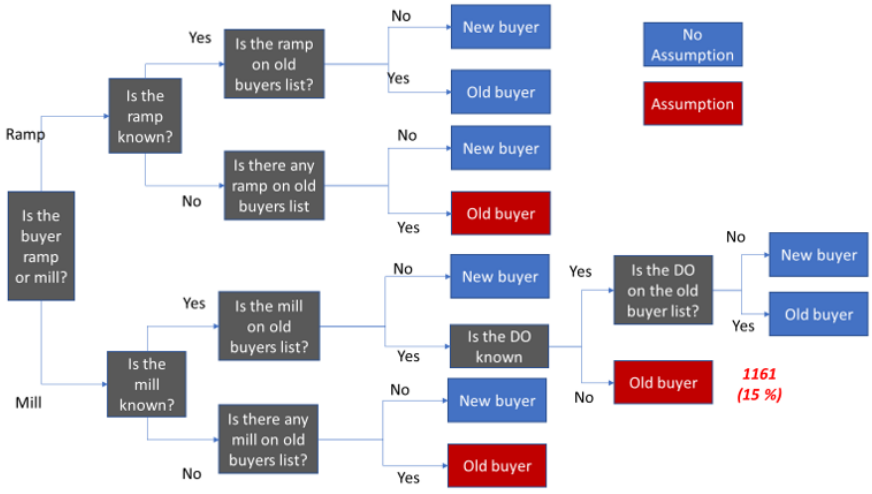


Figure 3-11: New buyer determination schematic

ported in the in-app survey and the 2018 field to calculate the counterfactual variables. For the special group robust case, we use all buyers in the *old_buyers_df* table.

The source of price, post grading price, and grading correction rate information is *counterfactual_price* table. To calculate the counterfactual price, we take the relevant price data based on the user_id, date, and relevant buyers (depending on the counterfactual case), and take the relevant statistics for the variables (e.g. the mean of the prices of the counterfactual buyers for mean counterfactual case, and the max of the prices of the counterfactual buyers for max counterfactual case). For example, if we want to estimate the counterfactual price for user_id 31 on March 1 2020 for mean counterfactual case, we first get the list of counterfactual buyers from the *old_buyers_df* table for user_id 31. Then, we take the price of every buyer in the list of counterfactual buyers, and take the average of the price as the counterfactual price. The calculation for grading correction rate and post grading price is done in similar manner.

The counterfactual revenue calculation for the mean counterfactual case is done by

multiplying the counterfactual price and post grading weight as presented in Equation 3.6. In the counterfactual max case, the revenue is calculated as the product of total weight and counterfactual post grading price, which is presented in Equation 3.7.

$$\begin{aligned} &\text{CounterfactualRevenueMeanCase (IDR)} \equiv \\ &\text{CounterfactualPriceMean (IDR/kg)} \times \\ &\text{PostGradingWeight (kg)} \end{aligned} \tag{3.6}$$

$$\begin{aligned} &\text{CounterfactualRevenueMaxCase (IDR)} \equiv \\ &\text{CounterfactualPriceMax (IDR/kg)} \times \text{PostGradingWeight (kg)} \end{aligned} \tag{3.7}$$

The detailed method for this calculation is presented in the *Notebook 5* in the appendices.

3.5 Statistical Method

In this section, we will describe the statistical methods used in this study.

3.5.1 t-Test

We used t-test to the perform difference in means analysis to test whether the means of two different population are the same and whether the impact of the treatment is significantly different than a certain value. In the difference in means analysis, we used an independent sample t-test. However as there is clustering in the data that we want to

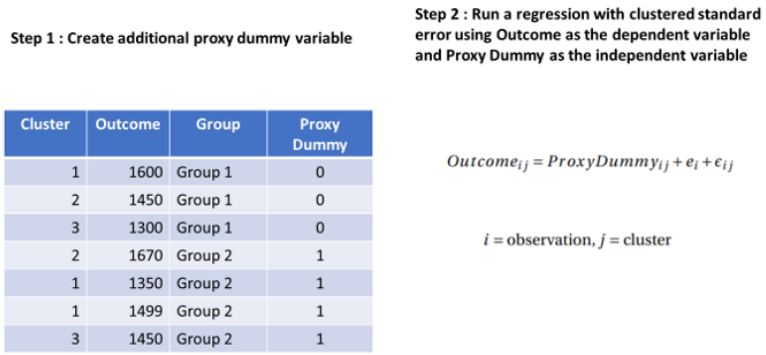


Figure 3-12: The procedure in performing difference in means analysis with clustered standard error

take into account, we perform the regression equivalent of the independent sample t-test. The graphical representation on how to achieve this is presented in Figure 3-12 on page 58. Let's assume that we start with data on the outcome of each observation in the two groups. To estimate the difference in means between the two groups, we follow a 2-step procedure. Firstly, we created an additional variable, *Proxy Dummy*, to indicate the group to which each observation belongs. Next we regress the *Outcome* on *Proxy Dummy* with clustered standard error on *Cluster*. The resulting regression coefficient of *Proxy Dummy* can be interpreted as the difference in means between Group 2 and Group 1. The corresponding t-statistics can be used to perform the hypothesis testing whether the difference in means between the two groups is statistically non-zero or not. This kind of transformation is also used in a typical regression adjustment analysis. But unlike regression adjustment we don't include any covariates, and hence this method does not address any confounding issues.

3.5.2 OLS Regression

In estimating the causal impact of our treatment, we used Ordinary Least Square (OLS) regression. The OLS regression will provide a consistent estimate for the estimand of interest when the conditional independence assumption or selection-on-observable holds [21]. For a regression specification in Equation 3.8, the estimated coefficient for the regressor can be expressed as in Equation 3.9, where $Cov(y_i, x_i)$ is the covariance of x_i and y_i and $Var(x_i)$ is the variance of x_i. This equation is also referred to as the regression anatomy [21].

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{3.8}$$

$$\beta_1 = \frac{Cov(y_i, x_i)}{Var(x_i)} \tag{3.9}$$

3.5.3 LASSO Regression

LASSO (Least Absolute Shrinkage and Selection Operator) is a penalized regression method commonly used to perform variable selection [37].. The penalization in the LASSO regression will provide a non-zero estimate only for variables that are highly predictive for the dependent variable. In this work, we used 10-fold cross validation to optimize the penalization term. This procedure allows us to select which covariates are predictive of the outcome variables.

3.5.4 Post Double Selection LASSO Regression

While single selection LASSO with cross validation allows us to select which covariates best explain the variation in the outcome variable, estimating the estimand of inter-

est requires that the unconfoundedness assumption holds. The unconfoundedness assumption states that all variables affecting the estimand of interest and the outcome variable are observed and can be controlled for, and after controlling for them, we will arrive at the unbiased causal estimate [21]. This assumption is relevant when we are discussing the factors driving the heterogeneity in treatment effect in Section 4.5.4, where we need to consider the possibility that location and time affect the estimation of marginal effect of upload order, number of local price availability and special price memberships. As an illustration, please consider Figure 3-13. We are interested to estimate the marginal effect of upload order (estimand of interest) on post grading price treatment effect (outcome variable). However, we are aware that the time of transaction and the location of the user affect both the treatment and outcome variable. For example, users who reside in certain villages may be more likely to have high upload order or realize higher post grading price treatment effect. Hence to control for this, we need to select for the relevant confounding variables to be included in regression specification. Selecting for only the relevant covariates, and hence block the grey paths, will allow us to meet the unconfoundedness assumption. To this end, we used Post Double Selection (PDS) LASSO regression, which (under several technical assumptions) is guaranteed to provide a consistent estimate for the estimand of interest [2].

To provide an illustration on why PDS LASSO works, for ease of exposition but without loss of generality, let's consider two regression specifications given in Equation 3.10 (long regression) and Equation 3.11 (short regression), where y is the dependent variable, S is the estimand of interest, and A is the confounding variable. Let's assume that the long regression is the true regression specification that will give the unbiased estimate of the regression coefficient for the estimand of interest, ρ. Omitting the confounding variable and running the short regression specification will give a biased estimate of the regression coefficient for the estimand of interest, ρ_s. The relationship

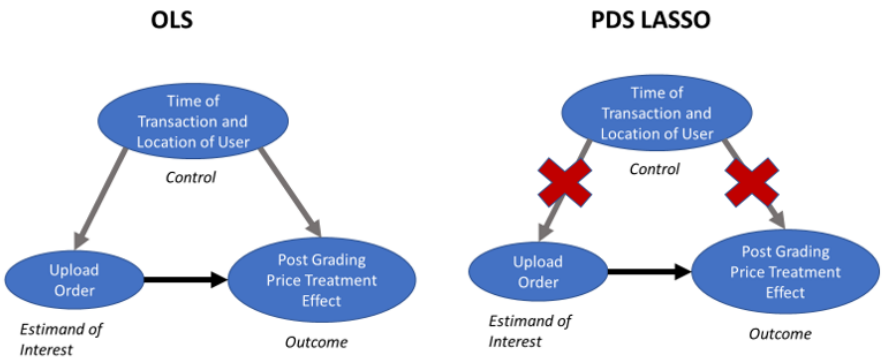


Figure 3-13: The directed acyclic graph model for the influence of upload order on post grading price treatment effect

between the biased estimate ρ_s and the unbiased estimate ρ is given by the Omitted Variable Bias (OVB) formula in Equation 3.12, where γ is the regression coefficient of A in the long regression (Equation 3.10), and δ_{As} is the regression coefficient for the confounding variable S in the regression specification in Equation 3.14. We will call the regression in Equation 3.14 the estimand of interest regression. By the regression anatomy formula, δ_{AS} can also be expressed as in Equation 3.15 (Angrist & Pischke 2008).

$$y_i = \alpha + \rho S_i + \gamma A_i + \epsilon_i \tag{3.10}$$

$$y_i = \alpha + \rho_s S_i + \epsilon_i \tag{3.11}$$

$$\rho_s = \rho + \overbrace{\gamma \delta_{As}}^{\text{Bias}} \tag{3.12}$$

$$\begin{aligned} \text{Bias} &= \rho - \rho_s \\ \text{Bias} &= -\gamma\delta_{As} \end{aligned} \tag{3.13}$$

$$A_i = \alpha + \delta_{AS} S_i + \epsilon_i \tag{3.14}$$

$$\delta_{AS} = \frac{Cov(A_i, S_i)}{Var(S_i)} \tag{3.15}$$

By rearranging Equation 3.12, the bias associated from the omission of the confounding variable S can be expressed as a product of two variables, γ and δ_{AS}, as presented in Equation 3.13. Intuitively, this means that a variable can confound the causal estimation if it affects both the dependent variable ($\gamma \neq 0$) and the estimand of interest ($\delta_{AS} \neq 0$). The problem with single selection LASSO is that it only picks the confounding variables by running the long regression in Equation 3.10. Single selection LASSO has the tendency to pick the variables with large γ magnitude, regardless of the magnitude of δ_{AS}. This will be a problem if there are confounding variables with large magnitude of δ_{AS}, but with low and non-zero value of γ. In this case, single selection LASSO will most likely fail to pick the confounding variable. To address this shortcoming, PDS LASSO is used. PDS LASSO, as its name implies, works by doing double selection on both the long regression in Equation 3.10 and the estimand of interest regression in Equation 3.14. The set of confounding variables picked by PDS LASSO is the union of the variables picked in the two selection process, the long regression in Equation 3.10 and the estimand of interest regression in Equation 3.14. By this double selection, PDS LASSO will pick the confounding variables that either affect the dependent variable or the estimand of interest. Hence the variables picked by PDS LASSO will provide the necessary control for the unbiased estimation for the estimand of interest.

3.5.5 Software

All the analysis in this work is done on Python and STATA. Data manipulation is done using Python pandas package [34]. Visualization is done using Python matplotlib and seaborn package [10], [30] . Statistical modelling is done using Python statsmodel package [41]. Lasso and PDS Lasso analysis is done using STATA lassopack program [1].

Chapter 4

Result

"It is better to be roughly right than precisely wrong."

John Maynard Keynes

4.1 Descriptive Statistics

The relevant descriptive statistics for this study are presented in Table 4.1, Table 4.2 and Table 4.3. Table 4.1 reports the breakdown of users by role in this study. From the table it can be seen that middle men & ramp managers sell at higher volume per transaction compared to farmers. Table 4.2 presents the comparison between the reported number of old buyers prior to treatment, total unique buyers transacted with after treatment, and total unique new buyers transacted with after treatment. It can be seen that there is no significant difference in the reported number of old buyers reported between farmer and middle man & ramp manager. However, we saw indications that middle man & ramp manager users have higher number of total buyers and total

Table 4.1: Descriptive statistics by role

	Number of Transactions	Number of Users	Post Grading Weight per Transaction
Farmer	1,456	233	2,717
Middle Man & Ramp Manager	6,477	280	4.923
All	7,933	513	4,624

Table 4.2: Number of old buyers reported & new buyer transacted by role

Statistics	User Role	Mean	Median	Min	Max	Std Dev
Old Buyers per User	Farmer	1.42	1	1	6	0.70
	Middle Man & Ramp Manager	1.82	2	1	5	0.89
	All Role	1.64	1	1	6	0.83
Total Buyer Transacted	Farmer	1.53	1	1	10	1.21
	Middle Man & Ramp Manager	2.50	1	1	28	2.91
	All Role	2.06	1	1	28	2.35
Total New Buyer Transacted	Farmer	1.42	1	1	6	0.98
	Middle Man & Ramp Manager	2.45	1	1	25	2.88
	All Role	2.04	1	1	25	2.38

new buyers transacted with after treatment. This indicates that middle man & ramp manager users make better use of the apps to access more buyers. Table 4.3 presents the statistics on the price reported by the sellers based on transaction type. It should be noted that ramp transactions have only been reported in the apps since June 2019, while mill transactions have been reported since the beginning of treatment in December 2018. This difference in timing makes head-to-head comparison between mill and ramp transaction prices difficult. In general, mills tend to offer better price compared to ramp, because the mill is a more downstream buyer compared to ramp. The average mill and ramp prices versus date are plotted in Figure 4-1 on page 67.

Table 4.3: Descriptive price statistics reported by the users (IDR/kg)

Statistics	Trans Type	Mean	Median	Min	Max	Std Dev
Price	Ramp Transaction	1,720	1,800	1,000	2,050	209
Price	Mill Transaction	1,600	1,630	905	2,200	285
Price	All Transaction	1,648	1,700	905	2,200	264
Post Grading Price	Ramp Transaction	1,653	1,714	940	1,990	195
Post Grading Price	Mill Transaction	1,525	1,548	862	2,101	274
Post Grading Price	All Transaction	1,576	1,623	862	2,101	254

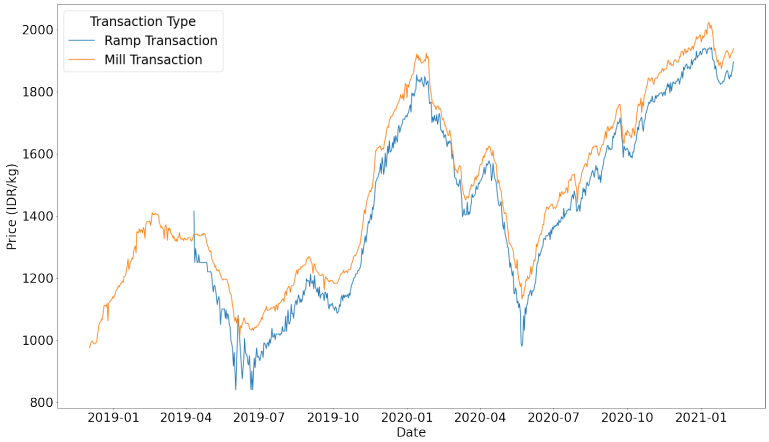


Figure 4-1: The evolution of average mill and ramp price reported in the platform

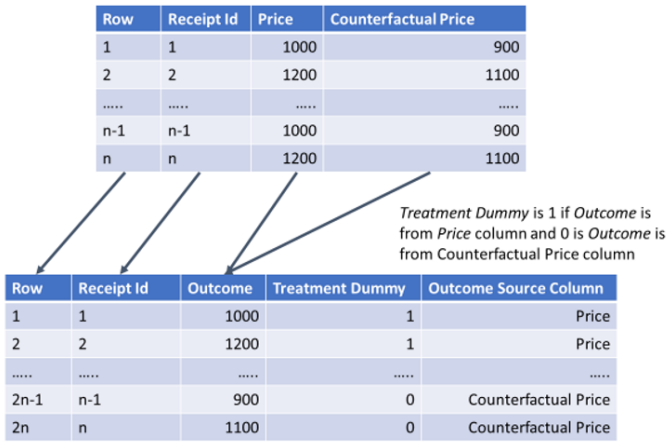


Figure 4-2: Schema for wide to long table transformation

4.2 Transaction Level Impact

In this work, price, post grading price, and revenue are the main variables of interest. On top of that, we also used six others dependent variables, listed in the Table 3.1 on page 36. For each dependent variable, we define the corresponding counterfactual as given in Table 3.1.

The treatment effect for each dependent variable is calculated as the difference between the realized variable and its counterfactual. To test the null the hypothesis that the treatment effect is zero, we opted to perform t-test with clustered standard error, which can be done by using the regression equivalence of 2-sample t-test. Instead of comparing the means of the realized variable and its counterfactual, we reshaped the data into a long form by stacking the realized and counterfactual variable in the same column. We then defined a treatment dummy variable to equal to 1 for realized variable, and 0 for its counterfactual. The schema for this table transformation is presented in Figure 4-2. We then regressed the outcome variable on the treatment dummy

variable using clustered standard error, as described by the regression specification in Equation 4.1. This transformation and calculation will yield the same result as 2 sample t-test, and it allows us to estimate the clustered robust standard error at user level. This kind of transformation is also used in a typical regression adjustment analysis.

One key feature of our analysis is that the data set for any regression is perfectly balanced, as for every observation row, there is a row with exactly the same covariates composition. This exact balance is a direct implication from the way we calculate the counterfactual variable, which is described in the previous section. This perfect balance implies that the inclusion of any covariates is unnecessary as every observation is paired with a perfect counterfactual. This case can also be viewed as an equivalent of perfect exact matching [39].

$$y_{ij} = \alpha + \beta \cdot x_{ij} + e_i + \epsilon_{ij} \tag{4.1}$$

$$x_{ij} = \begin{cases} 1 & \text{if x is realized variable} \\ 0 & \text{if x is counterfactual} \end{cases}$$

$$i = \text{user},\ j = \text{transaction}$$

Table 4.4 summarizes the transaction level analysis results. The results indicate that the access to market information on the platform increased the price realization by ~0.3% (p-value ~0.007), post grading price by ~0.4% (p-value ~0.006), price and post grading price viewed by ~2% (p-value < 0.001) , price and post grading price availability by ~7% (p-value < 0.001), and revenue by ~0.4% (p-value ~0.007). The detailed regression result is presented in Table 4.4. The transaction level analysis presents evidence that the treatment has a statistically non-zero and positive impact of price, post grad-

ing price, and revenue.

Table 4.4: Transaction Level Regression Result

Variable	Unit	Number of Clusters	Number of Transactions	Effect (Absolute)	Standard Error	P values	Effect (in %)
Price	IDR / kg	391	5673	5.212	1.936	0.007	0.32
Best Price	IDR / kg	394	5734	115.964	9.592	0.000	7.04
Best Price Viewed	IDR / kg	394	5719	32.218	4.309	0.000	1.96
Post Grading Price	IDR / kg	341	5312	7.008	2.550	0.006	0.45
Best Post Grading Price	IDR / kg	355	5456	115.853	8.147	0.000	7.38
Best Post Grading Price Viewed	IDR / kg	351	5412	34.534	3.369	0.000	2.20
Grade	%	347	5406	- 0.127	0.055	0.022	- 2.68
Revenue	IDR	320	5258	33,046	12,201	0.007	0.46

Note: Effect (in %) is calculated by dividing the Effect (Absolute) with the intercept and multiply with 100%

4.3 User Level Impact

We analysed the impact of treatment at the user level by aggregating the realized revenue that the user gets while using the app and comparing it with the estimated revenue that the users would have received in the absence of the app. We then defined a new variable DeltaRevenue% as the difference between the aggregate revenue and the counterfactual revenue, which is detailed in Equation 4.2.

$$\text{DeltaRevenue\%}_i = \frac{\sum_j \text{Revenue}_j - \sum_j \text{CounterfactualRevenue}_i}{\sum_j \text{CounterfactualRevenue}_j}.100\% \qquad (4.2)$$

$$i = \text{user},\ j = \text{transaction}$$

We performed a t-test to test the null hypothesis that the DeltaRevenue% variable is zero. The result of the analysis is summarized in Table 4.5. The result shows an aggregate per user revenue increase of 0.16%, with p-values of 0.165 for all roles, which indicates that the null hypothesis can't be rejected.

A subgroup analysis reveals a statistically non-zero result of DeltaRevenue% for middle man & ramp manager, with an estimated average of 0.3% with a p-value of 0.076. Middle men & ramp manager experience the highest DeltaRevenue%. The reason for this can be attributed the higher transaction frequency of the middle men & ramp managers. We will explain this in more detail in Section 4.5.1, when we discuss the seller's learning as the factor impacting heterogeneity in treatment effect. Another possible reason for this is that middle man & ramp manager sell at higher volume per transaction that allow that to travel more distance and receive better price because the transportation cost can be spread over larger volume.

Table 4.5: t-Test of Delta Revenue %

Role	Number of Users	Mean	SEM	p- Values
Farmer	151	-0.017	0.137	0.900
Middle Man & Ramp Manager	169	0.314	0.176	0.076
All	320	0.158	0.111	0.165

4.4 Temporal Variation in User Behavior

We expect that users will benefit from using the app by accessing information beyond their initial circle of buyers, and as they do so, they will be more inclined to transact outside of their initial circle of buyers. To further understand how the user transitions from their old buyers to the new buyers whose market information they access via the app, we calculated the cumulative new buyer fraction for each upload order.

Upload order is defined as the chronological sequence of dates for which the transaction receipt is reported by the user on the app, based on his history. A user can upload more than one transaction receipt per day. But if the two or more transaction receipts are reported on the same day, those transactions will have the same upload order. As illustration, for the tenth upload ($n = 10$), the cumulative new buyer fraction is defined as the total number of transactions involving a new buyer divided by the total number of transactions. The calculation is described in Equation 4.3.

The rolling average plot of fraction of cumulative new buyer versus upload order is presented in Figure 4-3 on page 75. The rolling average for fraction of cumulative new buyer is calculated with several window sizes: 50, 75, and 100. As an illustration, for a window size of 100, the subset $n-99$ in x-axis in the plot corresponds to the fraction of cumulative new buyers in the upload order $n-99$ to upload order n. The figure displays an increasing slope, indicating that as the users gain experience in the app, they are more inclined to transact with a new buyer. The fact that the three window

sizes all produce similar plots indicates that the analysis is not sensitive to the choice of window size.

$$\text{CumulativeNewBuyerFraction\%}_n = \frac{\sum_{i=1}^{n} \mathbb{1}(\text{Buyer}_i = \text{New Buyer})}{n} \times 100\% \tag{4.3}$$

As the users get more inclined to transact with new buyers the longer they use the app, we asked the question whether, in addition to expanding the circle of market interactions, the treatment effect also increases with user's learning. To this end, we performed a rolling regression analysis on post grading price, where we used a similar regression specification as the one used in Equation 4.1. The difference here is the subsetting of the data. We used a window size of 100 and limited the analysis to upload order less than 193 ($n < 193$), as it provides at least 20 clusters and sufficient statistical power to perform asymptotic analysis (Cameron et. al. 2008). Using this window size, for subset n, we include all the transactions where the upload order is between $n-99$ and n.

The result of this analysis is presented in Figure 4-4 on page 77, with the number of transactions and clusters involved in this analysis presented in Figure 4-5 page 79. The plot indicates that the treatment effect may increase with increasing upload order. To investigate whether this observation can be attributed to (1) learning effect (the user became savvier in using the apps), (2) increase in information quality in the apps (there is more price information available in the app as the app matures), or (3) both, we performed similar rolling regression analysis with best post grading price and best post grading price viewed as the dependent variables.

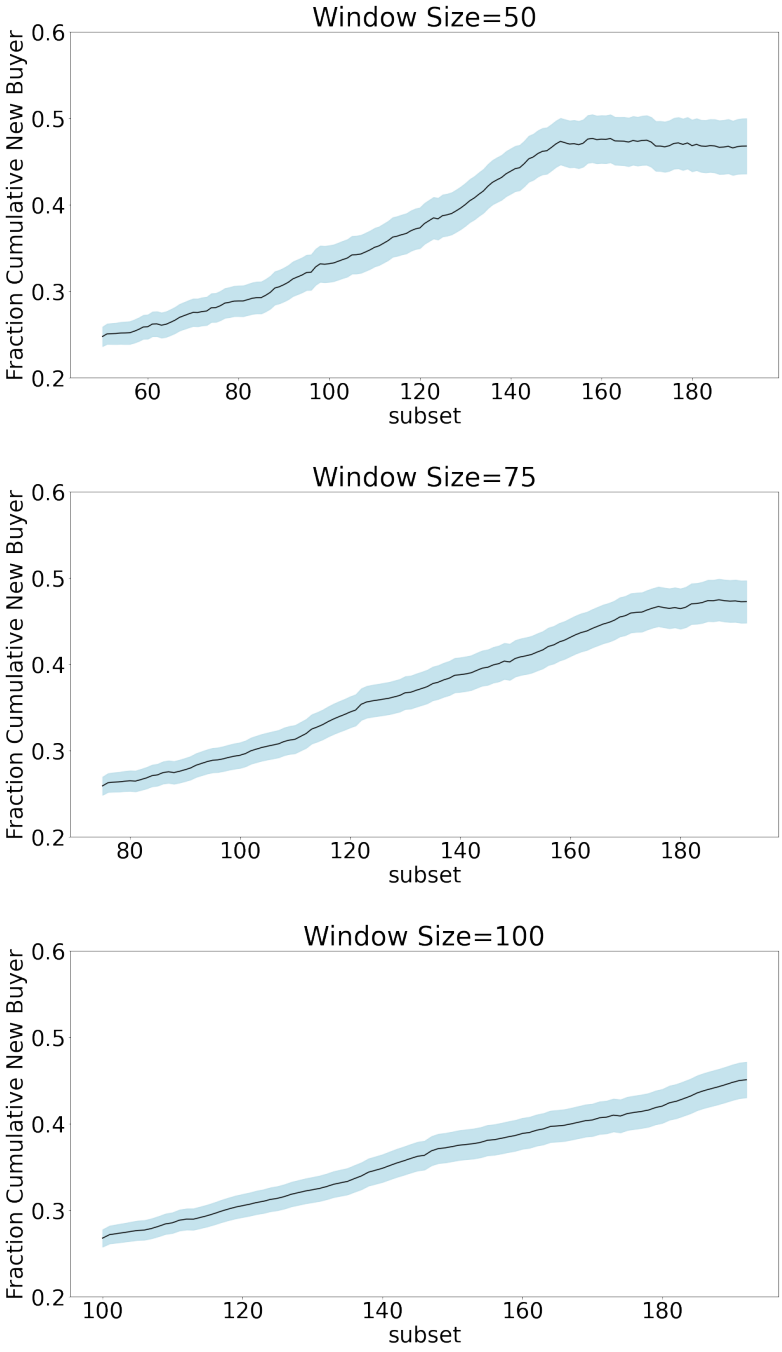


Figure 4-3: Fraction of cumulative new buyer versus upload order
Note : the light blue band indicates 95 % confidence interval

We observed a relatively decreasing slope and flat trend for the best post grading price viewed and best post grading price respectively, as can be seen in Figure 4-4 on page 77. The observation that the rolling regression trends on best post grading price viewed and best post grading price display a non-increasing slope corroborates the hypothesis that the increase in treatment effect is caused by the learning effect and not by the improvement on price availability on the platform.

To further estimate the slope, we regress the estimated effect on the subset. The estimated slope for this analysis is presented in Table 4.6. To account for the presence of positive auto-correlation in the residual, we ran OLS with Newey West standard error both with minimum number of lag (1) and maximum number of lag (91). All of the estimated slopes are statistically non-zero, with the estimated slope for post grading price treatment effect being greater than zero and the other two slopes less than zero. This corroborates the notion that the increase in the treatment effect over time is driven by seller's learning and not by the increase in price availability.

4.5 Heterogeneity in Treatment Effect

We performed heterogeneity analysis to investigate the factors driving the variation in user behaviours and outcome. Out of the available outcome variables, we focus our analysis on post grading price. We proposed the framework presented in Figure 4-6 on page 81. The hypothesis framework is built from the Rubin causal model discussed in Section 3.2.1, where the observed post grading price can be expressed as a sum of counterfactual post grading price and the treatment effect.

We hypothesized that the treatment effect is driven by two main factors, the price differential across buyers and the seller's decision making. In the absence of price differential across buyers, i.e. every buyer offers the same price, the treatment effect will

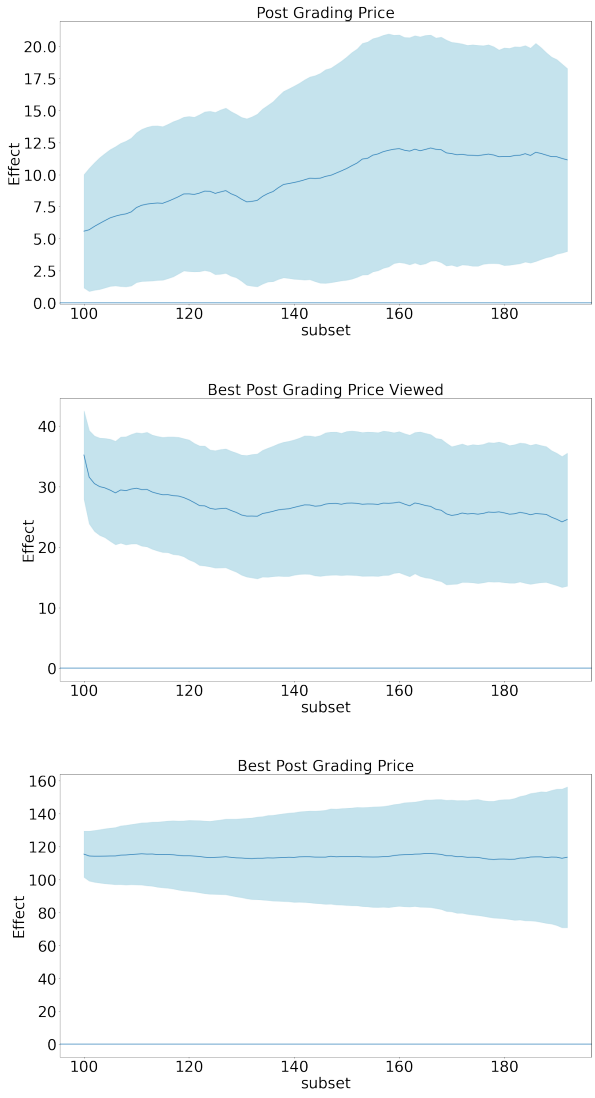


Figure 4-4: Rolling regression analysis
Note : the light blue band indicates 95 % confidence interval

Table 4.6: Regression result for the estimation of the slopes in Figure 4-4

Dependent Variable : Post Grading Price Treatment Effect			
	(1)	(2)	(3)
	OLS	OLS with Newey Standard Error Lag 1	OLS with Newey Standard Error Lag 91
subset	0.0651***	0.0651***	0.0651***
	(0.00296)	(0.00406)	(0.00686)
Constant	0.263	0.263	0.263
	(0.411)	(0.561)	(0.974)
Observations	93	93	93

Dependent Variable : Best Post Grading Price Viewed Treatment Effect			
	(4)	(5)	(6)
	OLS	OLS with Newey Standard Error Lag 1	OLS with Newey Standard Error Lag 91
subset	-0.0479***	-0.0479***	-0.0479***
	(0.00531)	(0.00660)	(0.00851)
Constant	33.95***	33.95***	33.95***
	(0.871)	(1.088)	(1.326)
Observations	93	93	93

Dependent Variable : Best Post Grading Price Treatment Effect			
	(7)	(8)	(9)
	OLS	OLS with Newey Standard Error Lag 1	OLS with Newey Standard Error Lag 91
subset	-0.0128***	-0.0128***	-0.0128***
	(0.00292)	(0.00398)	(0.00321)
Constant	115.9***	115.9***	115.9***
	(0.419)	(0.569)	(0.475)
Observations	93	93	93
Robust standard errors in parentheses			
*** p<0.01, ** p<0.05, * p<0.1			

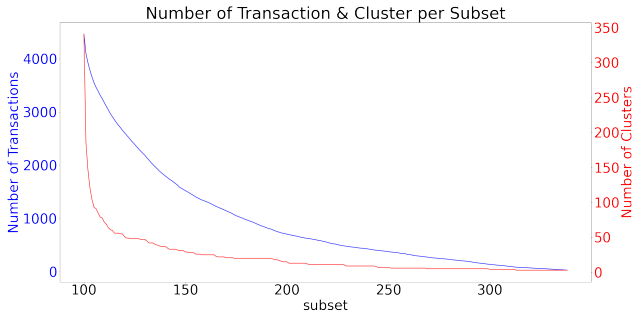


Figure 4-5: Number of transactions & clusters in rolling regression analysis

be zero. A price differential across buyers can be viewed as an in-addressable market fundamental. And hence, we don't focus our analysis on this bucket.

Even in the presence of a very high price differential across buyers, the treatment effect will still be zero if the sellers decide not to act on it. This is why we believe that the seller's decision making is another factor affecting the treatment effect. The seller's decision making can be viewed as something that we can aim to improve, and hence it becomes the focus of our analysis. The four main factors affecting the seller's decision making are: seller's learning, information availability, prior commitment of the sellers, and the ease of transportation.

- **Seller's Learning**. We want to test the null hypothesis that seller's learning has no significant effect on the post grading price treatment effect, with upload order as the proxy variable. We expect that even if the sellers are aware of the presence of better prices, there may still be some inertia to transact with a new buyer. But as the sellers gain more experience with the platform, the inertia may decrease. We tested this hypothesis by studying how the treatment effect changes with the number of sales transaction a user reported on the app. We approximate the number of sale transactions of a user by the number of sale transactions that

the user has uploaded onto the platform.

- **Local Information Availability**. We want to test the null hypothesis that local information availability has no significant effect on the post grading price treatment effect. We tested this hypothesis by examining the correlation between the magnitude of the treatment effect and the amount of local information available. We measure “local information” using a proxy for village proximity based on adjacency to the seller’s village. Ideally, we would measure the proximity between a seller and a specific buyer, using absolute distance between them. However, because we do not have this data, we instead use "village distance" as a substitute. Village distance between a seller and a buyer can be summarized as the minimum number of villages that the seller needs to travel to reach the buyer. More detailed explanation for this choice is given in section 4.5.2.

- **Prior Commitment to Sell to A Specific Buyer**.We want to test the null hypothesis that prior commitment has no significant effect on the post grading price treatment effect. Users belonging to special price membership usually have a commitment to sell to specific buyers, either to serve outstanding debt or to maintain goodwill. Hence, special price membership is chosen as proxy variable for prior commitment.

- **Transportation**.We want to test the null hypothesis that ease of transportation has no significant effect on the post grading price treatment effect. A seller with a better means of transportation (e.g. larger or better truck) may be better off in accessing new buyer. A potential relevant proxy variable for this is the type of truck that sellers have. In addition to that, the road infrastructure varies significantly across villages. Sellers in villages with better road infrastructure may be better

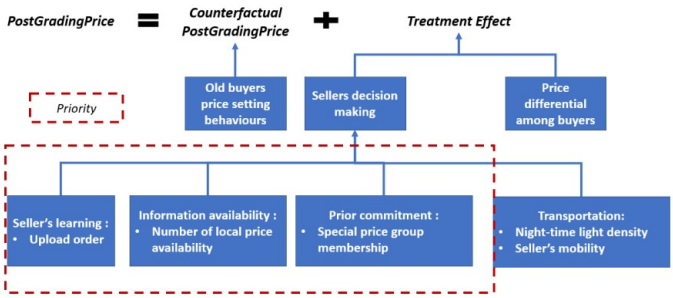


Figure 4-6: The hypothesis framework to study the factors affecting heterogeneous treatment effect

off when accessing a new buyer. A potential relevant proxy variable for this is the satellite data on the night-time light density in different villages. However, due to time and travel restriction, we do not have a reliable data source for these two variables. As such, we defer the analysis of this factor, and focus our effort on the first three factors.

4.5.1 Seller's Learning

In Section 4.4, we have presented an indication that seller's learning leads to an increase in treatment effect. In Section 4.3, we observed that middlemen and ramp managers exhibit a statistically non-zero positive DeltaRevenue% increase. One possible explanation for this could be that middlemen and ramp managers transact more frequently than farmer, hence they require less time to master the app. On average, middlemen and ramp managers report 1 transaction receipt every 22 days, compared to 26 days for farmer as presented in Table 4.7. The average upload period is calculated as the ratio between the difference between the last and first upload date, divided by the number of upload order between the last and first upload date excluding the first upload order. The calculation is presented in Equation 4.4. A more detailed discussion on this will be

Table 4.7: Average upload period (in days)

Role	Number of Users	Mean	Standard Error of Mean
Farmer	94	26.90	4.08
Middle Man & Ramp Manager	145	22.12	2.60
All Role	239	24.00	2.25

described in Section 4.5.4, when we discuss the regression analysis.

$$\text{AverageUploadPeriod}_j = \frac{\text{Number of Days Between Last and First Upload}_j}{\text{Number of Upload Order Between Last and First Upload}_j} \tag{4.4}$$

As an additional analysis, we performed subgroup tests between transactions with high upload order and transactions with low upload order. We defined transactions with high upload order as those transactions whose upload order is higher than or equal to the median upload order (upload order = 33). We conducted a difference in means test using clustered t-test on two outcome variables : post grading price treatment effect and new buyer. Post grading price treatment effect is defined as the difference between the post grading price that the sellers received and the post grading price that they would have received in the absence of the treatment. New buyer is a binary variable indicating whether the user is transacting with a new buyer (1) or not (0). The merit of this method lies on its simple but rigorous approach and the lack of subjective parameter selection (the threshold is simply the median). One drawback of this analysis is that it assumes that the 2 groups are balanced in covariates composition and comparable to each other. In the presence of confounding, this is not the case. To address this issue, we perform regression in Section 4.5.4.

The result of this test is presented on Table 4.8. We found evidence that transactions

Table 4.8: The difference in means analysis for upload order

Variable	Difference in Means	p-Value
Post Grading Price Treatment Effect	5.8027	0.096
New Buyer Rate	0.1416	0.045

in the high upload order subgroup have a higher post grading price treatment effect of 5.8 IDR / kg or 80% of the average treatment effect (significant at <10% confidence level) and higher affinity to transact with a new buyer of 14% (significant at <5% confidence level).

4.5.2 Local Information Availability

Not all information on the platform is relevant to all sellers, and not all information relevant to all sellers is on the platform. Specifically, when sellers make the decision of where to sell, the farmers/sellers will take into consideration the offer price of the buyers within certain distance, rather than the entire buyer population. Because the platform does not necessarily include all the buyers in all regions relevant to sellers, it is important to correct for this.

Because we do not have reliable data on road distance, we created a proxy variable called “village distance”. Village distance is calculated by creating a network connectivity village model. Using the polygon shape file of the village, we model the village as a network with each village as a node. For each pair of villages, if the polygons of the villages touch each other, then the 2 villages are connected nodes that share an edge between them. The village distance between 2 villages is defined as the minimum number of edges connecting them. As an example, consider the village BATANG GANSAL BELIMBING in Figure 4-7. The polygon of the village BATANG GANSAL BELIMBING is colored in red. Each of the orange polygons touches the red polygon, and hence in the

village network model, each of the orange polygons is connected with an edge to the red polygon. Consequently, the village distance between the red polygon to any of the orange polygons is 1. On other hand, none of the grey polygons directly touches the red polygon. However, if a user travels from any of the grey polygon to the red polygon, he can do so by crossing one of the orange polygon. In the network model, this means that the shortest path between any of the grey polygons to the red polygon is 2 edges, because if one starts from the grey node, one can reach the red by hopping onto at least 1 orange node. Hence, the village distance between any of the grey polygons to the red polygon is 2.

Using the village distance method, we studied the page browsing and transaction behaviours of the users. We found that more than 97% of transactions happened when the distance between buyer and seller was less than or equal to 5 villages away (5 edges in the village network model). More than 87% of page views by the user in the apps are of buyers no more than five villages distant. The relevant histogram is presented in Figure 4-8 on page 86. Based on this, we defined local information availability as the number of unique buyers located less than or equal to 5 village distant from the user whose price information is available is available on the app.

Having established that a five village distance is a reasonable threshold for relevant local information availability, we studied whether transactions that happened when the seller has high local information availability has different post grading price treatment effect compared to those with low local information availability. To do this we partitioned the transaction into two parts. The first part, the transaction with high local information availability, refers to the transaction that happened when the corresponding seller has local information availability higher than or equal to the median local information availability (32 prices). The second set of transactions are the transaction with low local information availability, which refers to the transaction that happened

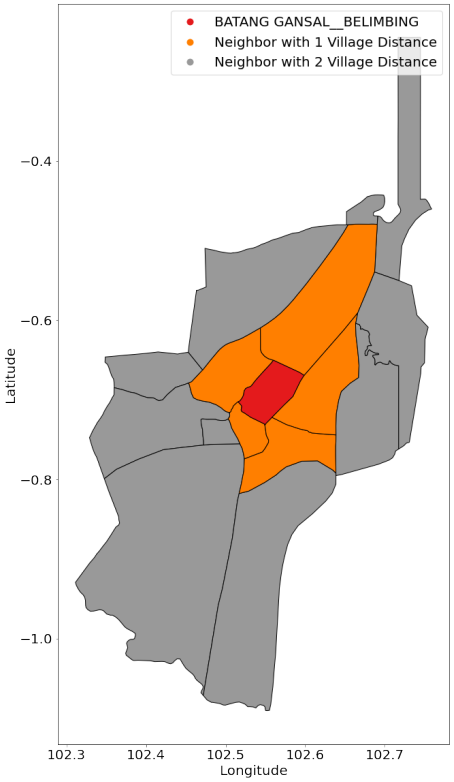


Figure 4-7: An example of village distance calculation

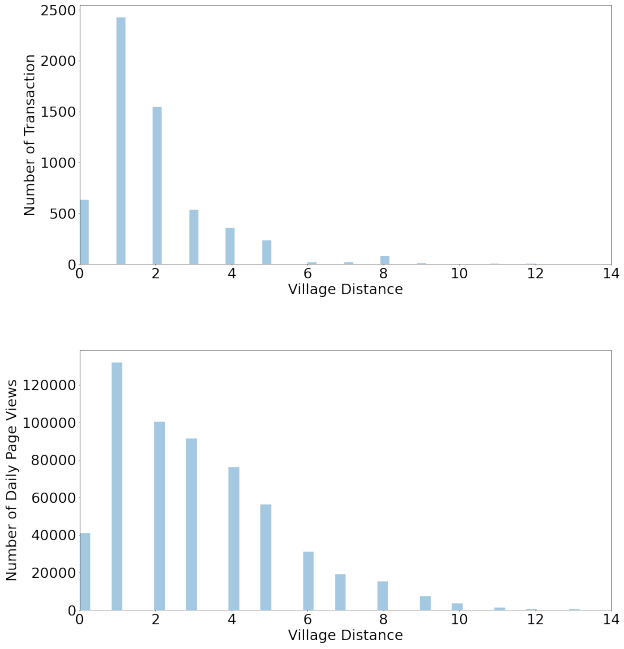


Figure 4-8: The histogram of transaction and page view based on the village distance between user and buyer

Table 4.9: The difference in means analysis for local information availability

Variable	**Difference in Means**	**p-Value**
Post Grading Price Treatment Effect	0.348	0.923
New Buyer	0.160	0.039

when the corresponding seller has local information availability lower than the median local information availability. We then conducted difference in means t-test between the two subsets, using two outcome variables: post grading price treatment effect and new buyer.

The result of the analysis is presented in Table 4.9. The result indicates that information availability does not necessarily increase the post grading price treatment effect, but it leads to an increase in new buyer rate. One explanation for this that a seller may discover new buyers who are located closer than his old buyers. Sellers can then choose to transact with those new buyers to save transportation costs, even if those new buyers offer lower prices. In this case, the seller may receive a lower price and hence negative treatment effect, but he still benefited from the reduced travelling distance. The result estimated that instances with high local information availability have 16% higher chance of new buyer transaction.

4.5.3 Prior Commitment

To study the effect of prior commitment on new buyer transaction rates and the post grading price treatment effect, we a conducted similar difference in means test as in Section 4.5.2.

The result of the analysis is presented in Table 4.10. Firstly we found that the difference in post grading price treatment effect is negative, but not significant. While insignificant, the sign of the estimate is consistent with what we intuitively expect. The

Table 4.10: The difference in means analysis for prior commitment

Variable	**Difference in Means**	**p-Value**
Post Grading Price Treatment Effect	-4.173	0.334
New Buyer	-0.257	0.003

users with special price membership will have less flexibility in choosing which buyer to sell to. Consequently their post grading price treatment effect is expected to be lower compared to the other group. It is to be noted that these results do not contradict the anecdotal information that special price members tend to get the best price in the market. A seller can get the best price in the market, but if it is an old buyer transaction, the post grading price treatment effect is 0 by definition.

Additionally, we found that sellers with special price membership have a lower new buyer transaction rate (significant at <1% significance level). Thus, sellers who have special relationships with a buyer (e.g. due to debt or special price rates) are significantly less likely to transact with a new buyer, which is consistent with our expectation. The result suggests that users with special price membership have 26% less affinity to transact with a new buyer (significant at <1% significance level).

4.5.4 Regression Analysis

In the preceding subsection, we have performed difference in means analysis to explore the drivers of treatment effect heterogeneity. While difference in means analysis enables us to perform rigorous and objective analysis, it does not yield an unbiased estimate for the marginal of effect of the because it does not correct for other potentially important covariates. To provide an unbiased estimate of the marginal effect of of the proxy variables affecting the post grading price treatment effect, we performed regression analysis following the regression specification given in Equation 4.5, using

two outcome variables : post grading price treatment effect and new buyer rate. Apart of the three proxy variables, we also include three control variables, the village dummy, sub-district dummy, and the month dummy, to control for spatial and temporal variations. To take into account the correlation between transaction from the same user, we incorporated two error terms in the regression specification, one for the idiosyncratic error of the user and one for the idiosyncratic error of the transaction. The village dummy variable is a dummy indicating whether the user dwells in the given village (1) or not (0). Similarly, the sub-district dummy variable is a dummy indicating whether the user dwell in the given sub-district (1) or not (0). Due to multi-collinearity, sub-district dummy is only used in penalized regression analysis. The month dummy is a dummy variable indicating whether the transaction happen in the given month (1) or not (0). To take into consideration the presence of clustering in our data set, inference is done using clustered robust standard error at user level.

$$\begin{aligned} \text{Outcome}_{ij} = \text{Covariate}_{ij} + \text{VillageDummy}_i + \\ \text{SubDistrictDummy}_i + \text{MonthDummy}_{ij} + e_i + \epsilon_{ij} \end{aligned} \tag{4.5}$$

$$i = \text{user},\ j = \text{transaction}$$

There are 116 village dummy variables and 27 month dummy variables. The inclusion of village dummies and month dummies may result in the loss of degree of freedom, which may lead to under-rejection. To address this issue, we used 10- fold cross-validation Lasso (CV Lasso) to select the relevant variables. Out of 3 covariates that we explored, CV Lasso only picked upload order together with a subset of village dum-

mies and month dummies as the relevant variables for post grading price treatment effect. This result indicates that upload order has the most predictive power among the 3 covariates considered, and that the other 2 covariates have less predictive power compared to the selected village dummy and temporal dummy. This result is consistent with the result from the preceding subsections that indicate local information availability and special price group membership do not affect the post grading price treatment effect.

After identifying the relevant variables, we ran Post Lasso OLS estimation to test the hypothesis that the selected covariate, upload order, has a non-zero effect on post grading price treatment effect. We used 2 different methods, OLS Post CV Lasso and OLS Post Double Selection Lasso (PDS Lasso). PDS Lasso is a newly developed technique which suitable for the unbiased estimation of estimand of interest (in this case, upload order), as explained in Section 3.5.4. The OLS Post CV Lasso is presented as comparison. The results of the regression are presented in Table 4.11. The regression results are consistent with the result from upload order rolling regression, indicating seller learning as the main driver for the post grading price treatment effect. We found no evidence that local information availability and special price membership affect post grading price treatment effect. The regression results estimated that a one unit increase in upload order will increase the post grading price treatment effect by 0.1 IDR or around 1.5% of the average treatment effect. The highest upload order in the subset of data used for heterogeneity analysis is 309, with an estimated treatment effect close to 26 IDR, or around 4 times larger than the average treatment effect. We will discuss how treatment varies with upload order in subsection 4.5.5.

We ran a similar regression with new buyer rate as the dependent variable. The 10-fold CV Lasso procedure picked the number of local price availability and special price dummy, together with a subset of village and temporal dummy as the most pre-

dictive covariates. The results of the regression are summarized in Table 4.12. As we are dealing with binary dependent variable, we used a logit model and estimated the average marginal effect (AME) of the covariates. The post CV Lasso estimates indicate that number of local price availability (number of buyers within 5 village distance from the user whose price information is available on the app) exhibits a statistically significant impact on the new buyer transaction rate. The logit AME and OLS estimated that a one unit increase in price availability within 5 village distance from the seller's village will lead to a 0.5 - 0.7% increase in the affinity to transact with new buyer (column (7) and column (8) on Table 4.12). The PDS Lasso method estimates that a one unit increase in price availability within 5 village distance from the seller's village will lead to a one percent increase in the affinity to transact with a new buyer, significant at the <1% confidence level.

Table 4.11: The regression result of the heterogeneity regression for dependent variable

Dependent variable : post grading price treatment effect								
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**
	OLS	**OLS**	**OLS**	**OLS**	**OLS**	**OLS**	**OLS Post CV Lasso**	**OLS PDS Lasso**
Upload Order	0.111**			0.0832			0.0741*	0.104**
	(0.0488)			(0.0533)			(0.0392)	(0.0494)
Number of Local Price Availability (5 Villages)		0.0149			-0.00992			
		(0.185)			(0.211)			
Special Price Member Dummy			-2.615			-0.555		
			(4.196)			(5.297)		
Village Dummy	No	No	No	Yes	Yes	Yes	Yes	Yes
Sub-district Dummy	No	No	No	No	No	No	Yes	Yes
Temporal Dummy	No	No	No	Yes	Yes	Yes	Yes	Yes
Observations	4,767	4,767	4,767	4,767	4,767	4,767	4,766	4,767
Cluster robust standard errors in parentheses								
*** p<0.01, ** p<0.05, * p<0.1								

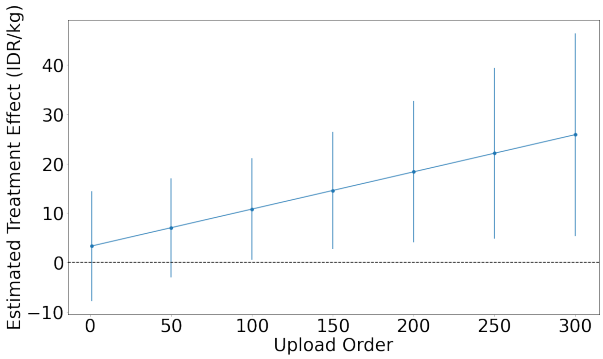
Table 4.12: The regression result of the heterogeneity regression for new buyer dependent variable

Dependent variable : new buyer									
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**
	Logit AME	**Logit AME**	**Logit AME**	**Logit AME**	**Logit AME**	**Logit AME**	**Logit AME Post CV Lasso**	**OLS Post CV Lasso**	**OLS PDS Lasso**
Upload Order	0.00130*			-7.53e-05					
	(0.000708)			(0.000599)					
Number of Local Price Availability		0.00681*			0.00469**		0.0073***	0.0066**	0.011***
		(0.00374)			(0.00235)		(0.0025)	(0.00259)	(0.0025)
Special Price Member Dummy			-0.109			-0.0964	-0.095	-0.090	-0.033
			(0.0945)			(0.125)	(0.087)	(0.090)	(0.093)
Village Dummy	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Sub-district Dummy	No	No	No	No	No	No	Yes	Yes	Yes
Temporal Dummy	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Observations	4,767	4,767	4,767	4,331	4,331	4,331	4,344	4,767	4,767
Cluster robust standard errors in parentheses. AME = Average Marginal Effect									
*** p<0.01, ** p<0.05, * p<0.1									

4.5.5 Subgroup Analysis

The preceding analysis on the heterogeneity in treatment effect has identified two main factors affecting the heterogeneity in treatment effect: the seller's learning curve and the number of prices in close proximity. To illustrate how post grading price treatment effect varies with the learning of the seller, we estimated the post grading price treatment effect for hypothetical users in BATANG GANSAL BELIMBING in January 2021, who have different levels of learning, as measured by upload order. We used the Post CV Lasso model fitted in the column (7) in Table 4.11 to estimate the post grading price treatment effect. We chose to use the single selection Lasso model because the single selection Lasso model performs better at predicting the dependent variable, while the PDS Lasso model is better at estimating the coefficient for the estimand of interest (upload order).

The result of this analysis is presented in Figure 4-9. The model predicted that a more experienced users with high upload order (upload order = 300, treatment effect = 25.90 IDR/kg) is estimated to have 7 times larger treatment effect compared to those with less experience (upload order = 1, treatment effect = 3.35 IDR/kg). One limitation of this analysis is that it is based on a linear functional form, which provides the best linear predictor of the marginal effect of upload order in the range of 1 to 309. Based on our analysis in Section 4.4, the influence of upload order on post grading price treatment effect may likely follow a non-linear trend with saturation beyond certain upload order.



Note: Vertical lines indicate 95% confidence interval

Figure 4-9: The difference in treatment effect among users with different level of learning

4.6 Sensitivity Analysis

As has been explained in Section 3.2.1, in the estimation of counterfactual outcomes, we rely on the assumption that, in the absence of the treatment, the set of buyers to whom the user can sell would remain the same. The set of counterfactual buyers is inferred from the survey administered to the users. In the preceding analysis, we estimate the counterfactual outcome as the average of the user's counterfactual buyers set. To test the robustness of our estimation, we perform two sensitivity analyses. In the first sensitivity analysis, we estimated the counterfactual outcome as the best potential outcome (highest price) that the user could have received in the absence of the treatment. The motivation for doing this is to investigate whether we can still maintain the same conclusion if we take a more conservative estimate of the treatment effect. We refer to this sensitivity analysis as the conservative case.

The second sensitivity analysis is motivated to investigate whether our conclusion still stands if the inferred counterfactual buyer set is incomplete. This can happen if there is a buyer with whom the user has a prior relationship, but the user failed to re-

port the buyer in the survey. Based on our understanding of the field situation, we believe that if a user belongs to a special group of a buyer, then it is likely that the user and the buyer have been acquainted before treatment. If this is the case, then the corresponding transaction between the user and the buyer should not be classified as a new buyer transaction. Hence, in the second sensitivity analysis, we expand the counterfactual buyers set to include all the buyers with whom the users are members of the same special group. We refer to this sensitivity analysis as the special group robust case.

The results of our sensitivity analysis for transaction level regression is presented in Table 4.13 and Table 4.14. For the conservative case, we found that the estimate for post grading price treatment effect is still significant at 95% confidence interval. The magnitude of the treatment effect estimate for price, post grading price and revenue are lower compared to the estimate in the average case scenario. The estimated treatment effect for price is ~ 0% (p-values ~ 0.977), the treatment effect for post grading price is ~ 0.32% (p-values ~ 0.038), and the treatment effect for revenue is ~ 0.02% (p-values ~ 0.921). For special group robust case, both the magnitude of the estimate and the statistical significance are in close agreement with the average case. We observed that the treatment effect for price is ~ 0.3% (p-values ~ 0.013), the treatment effect for post grading price is ~ 0.4% (p-values ~ 0.011), and the treatment effect for revenue is ~ 0.4% (p-values ~ 0.008).

Table 4.13: Transaction level result for max counterfactual case

Variable	Unit	Number of Clusters	Number of Transactions	Effect (Absolute)	Standard Error	P values	Effect (in %)
Price	IDR / kg	391	5673	0.064	2.206	0.977	0.00
Post Grading Price	IDR / kg	341	5312	5.037	2.434	0.038	0.32
Revenue	IDR	320	5258	1,179	11,901	0.921	0.02

Note: Effect (in %) is calculated by dividing the Effect (Absolute) with the intercept times 100%

Table 4.14: Transaction level result for special group robust case

Variable	Unit	Number of Clusters	Number of Transactions	Effect (Absolute)	Standard Error	P values	Effect (in %)
Price	IDR / kg	399	5796	4.710	1.894	0.013	0.29
Post Grading Price	IDR / kg	349	5448	6.426	2.518	0.011	0.41
Revenue	IDR	327	5349	32,021	11,997	0.008	0.44

Note: Effect (in %) is calculated by dividing the Effect (Absolute) with the intercept times 100%

We also studied how our rolling regression analysis on the impact of upload order on the treatment effect fares under the two sensitivity analysis. The results of the rolling regression for conservative and special group robust case are presented in Figure 4-10. The plot for both cases do not deviate too much from average case result, indicating that the conclusion from the average case analysis is still valid.

Our sensitivity analysis for transaction level analysis and rolling regression does not show much deviation from the default analysis. This indicates that the default case analysis is not sensitive to the relaxation of the identification assumptions. As such, we opted not to perform further sensitivity analysis.

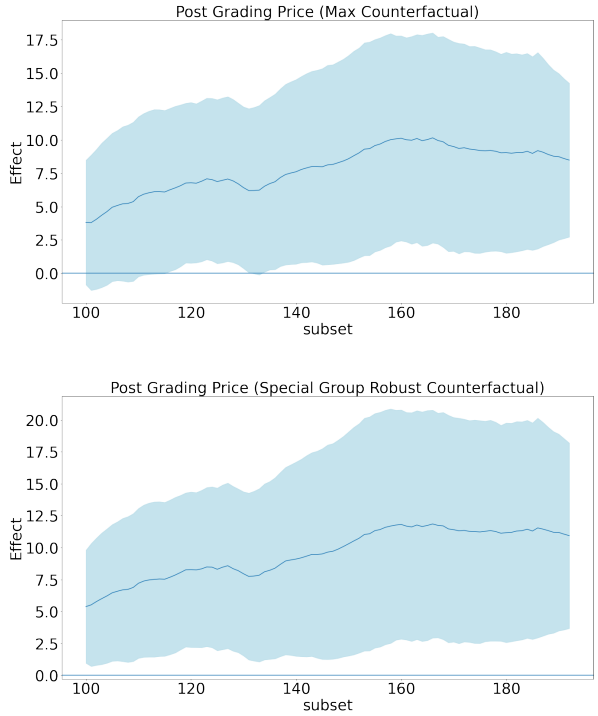


Figure 4-10: Sensitivity analysis of rolling regression
Note : the light blue band indicates 95 % confidence interval

Chapter 5

Conclusions and Future Work

"New knowledge is the most valuable commodity on earth. The more truth we have to work with, the richer we become."

Kurt Vonnegut, Cat's Cradle

We started this document by presenting the motivation in pursuing this research. Improvement in smallholder farmer productivity and welfare is of particular interest in the field of international development, driven by the concerns for food security and sustainable growth. In Chapter 2, we identified the current gap in the literature and how our work will seek to close this gap by asking two separate but related research questions. The first research question is whether an information crowd-sourcing platform has a significant impact on the welfare of the users. The second research question is what are the factors affecting the heterogeneity of the benefits of using a crowd-sourcing platform. Our ability to pursue these research questions is due to the unique data that we received from our partner which provides transaction level granularity and enables perfect matching of counterfactual, as explained in Chapter 3. In the same

chapter, we also presented the identification strategy and the methodology that we employ to uncover causal effect from the data.

We presented our answer to the research questions in Chapter 4. We presented evidence that the use of the app leads to an increase in the number of buyers that the users transact with. This indicates that the users use the platform to expand their social circle, allowing them to access more market information than what they otherwise would have. This improvement in market access allows them to realize a better price for their produce. We observed statistically significant increases in 3 measures of welfare : price (0.3%), post grading price (0.4%), and revenue (0.5%).

We investigated the driver for heterogeneity in treatment effect, with the hypothesis that treatment effect is driven primarily by the price differential among buyers and seller's decision making. We viewed the former as an unaddressable market fundamental, and focused on the latter. We hypothesized that the seller's decision making is driven by three factors : the seller's learning, local information availability, the seller's prior commitment to sell to specific buyers, and ease of transportation. Due to data and time constraints, we focused on the first 3 factors and deferred the analysis on ease of transportation. Each of these 3 factors was studied through the use of proxy variables; upload order for seller's learning, number of price quotes with 5 village distance for local information availability, and special price membership for prior commitment. We focus our analysis on two outcome variables, post grading price treatment effect and new buyer rate.

We employ difference in means analysis to study the presence of heterogeneity across the proxy variables. To control for other relevant covariates, we employ 10-fold CV Lasso and PDS Lasso. CV Lasso was used to select for the covariates that best predict the variation in the dependent variables, while PDS Lasso was used to provide a consistent estimate for the marginal effect of the variable of interest.

Difference in means and CV Lasso analysis indicated that local information availability and prior commitment were the main factors affecting the new buyer rate. Based on the data that we currently have, we only found evidence for statistically non-zero marginal effect for number of price availability with 5 village distance, but not for special price membership.

Difference in means analysis and CV Lasso indicates that the seller's learning is the main factor impacting the heterogeneity in the post grading price treatment effect. Additionally, we also found evidence that upload order exhibits statistically non-zero effect on post grading price treatment effect.

Non farmer intermediaries (middlemen and ramp managers) exhibit a statistically significant increase in revenue. Non farmer intermediaries also experience a higher increase in revenue compared to farmers. We attribute this difference to the higher transaction frequency of the non farmer group, which allows them to master the apps faster than their farmer counterpart. Another factor that may explain the higher benefit experienced by the non farmer group is the that non farmer group sells at a higher volume per transaction. Perhaps this higher volume allows them to travel greater distant to receive better prices as the transportation cost can be spread over higher volume. Linear regression modelling estimated that more experienced users can have up to 7 times higher treatment effect compared to less experienced users.

Due to the difficulty in estimating the transportation cost and travelling distance, our analysis focused exclusively on the top-line impact of the intervention (price and revenue). As such, if a user uses the app primarily to discover a closer buyer, our analysis does not account for this. Furthermore, if a user opted to transact with a closer new buyer offering lower price, our analysis yields a negative treatment effect, when the user actually benefits from the app in the form of a lower travelling distance. This indicates that our estimate of the causal impact of the intervention is a conservative

estimate compared to the actual impact of the intervention.

The practical implication of these findings is that it serves as proof of concept that an information crowd-sourcing platform can be a more cost effective solution to improve access to market information in an informal agricultural value chain. In comparison to a labor intensive agricultural extension model, an information crowd-sourcing platform has a lower cost structure. Perhaps the most significant cost for the app-based approach would be the user acquisition cost, which is incurred upfront. Our platform partner employs several field workers to establish initial operation in the village. But more than half of the users in the platform joined the platform organically, indicating the user acquisition cost is only incurred in the initial phase. Once the field operation reaches certain steady state level, potential users will organically joins the platform.

There are two potential future works that can be built on top of this work. Firstly, due data limitation and travel restriction, we haven't fully explored the underlying cause of geographical difference in treatment effect. Future studies can seek to investigate how villages differ from one another, and why some villages exhibit higher treatment effect than other. Specifically, a new study with better granularity for distance data, seller means of mobility, and road infrastructure will help to shed light on the travelling preference of the seller.

Secondly, mobile based interventions also have the potential to remove human biases often found in human-based intervention. It has been previously observed that extension workers tend to benefit male and more affluent farmers [15], [24]. Future studies can seek to study whether mobile based interventions exhibit similar biases or not.

Bibliography

[1] Ahrens A., Hansen C., and Schaffer M. LASSOPACK: Stata module for lasso, square-root lasso, elastic net, ridge, adaptive lasso estimation and cross-validation. Statistical Software Components, Boston College Department of Economics, February 2018.

[2] Belloni A., Chernozhukov V., and Hansen C. Inference on treatment effects after selection among high-dimensional controls. *The Review of Economic Studies*, 81(2):608–650, April 2014.

[3] Camacho A. and Conover E. The impact of receiving price and climate information in the agricultural sector. *Universidad de los Andes Department of Economics Research Paper Series*, 40, 2010.

[4] Goyal A. Information, direct access to farmers, and rural market performance in central india. *World Bank Policy Research Working Paper No. 5315*, 2010.

[5] Cunguara B. and Moder K. Is agricultural extension helping the poor? evidence from rural mozambique. *Journal of African Economies*, 20:562–595, August 2011.

[6] Rubin D. B. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688–70, 1974.

[7] Larochelle C., Alwang J., Barrera V.H., and Andrade J.M.D. Did you really get the message? using text reminders to stimulate adoption of agricultural technologies. *The Journal of Development Studies*, 55:548–564, 2017.

[8] Parker C., Ramdas K., and Savva N. Is it enough? evidence from a natural experiment in india's agriculture markets. *Management Science*, 62(9):2457–2764, September 2016.

[9] Pew Research Center. Smartphone ownership is growing rapidly around the world, but not always equally.

[10] Hunter J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[11] de Zegher J. and Lo I. Crowdsourcing market information from competitors. *Working Paper*, 2020.

[12] Beza E., Steinke J., van Etten J., Reidsma P., Fadda C., Mittra S., Mathur P., and Kooistra L. What are the prospects for citizen science in agriculture? evidence from three continents on motivation and mobile telephone use of resource-poor farmers. *PLoS One*, 12(5):e0175700, May 2017.

[13] Nakasone E. The role of price information in agricultural markets: Evidence from rural peru. *PhD Theses*, 2014.

[14] Schenk E. and Guittard C. Towards a characterization of crowdsourcing practices. *Journal of Innovation Economics and Management*, 1(7):93–107, 2011.

[15] Kondylis F., Mueller V., Sheriff G., and Zhu S. Do female instructors reduce gender bias in diffusion of sustainable land management techniques? experimental evidence from mozambique. *World Development*, 78:436–449, 2016.

[16] Rapsomanikis G. The economic lives of smallholder farmers : An analysis based on household data from nine countries. *FAO Report*, 2015.

[17] World Development Indicators Washington D.C. : World Bank Group. World development indicators 2016.

[18] World Resource Institute. Evaluating indonesia's progress on its climate commitments.

[19] Aker J. Information from markets near and far: Mobile phones and agricultural markets in niger. *American Economic Journal: Applied Economics*, 2:46–59, July 2010.

[20] Aker J. and Fafchamps M. Mobile phone coverage and producer markets: Evidence from west africa. *The World Bank Economic Review*, 29:262–292, 2015.

[21] Angrist J. and Pischke J. *Mostly Harmless Econometrics: An Empiricistís Companions*. Princeton University Press, 2008.

[22] Svensson J. and Yanagizama D. Getting prices right: The impact of the market information service in uganda. *Journal of the European Economic Association*, 7(2), 2009.

[23] Hernandez Aguilera J.N., Mauerman M., and Osgood D. Playing to adapt: Crowdsourcing historical climate data with gamification to improve farmer's risk management instruments. *Working Paper*, 2020.

[24] Saito K., Mekonnen H., and Spurling D. Raising the productivity of women farmers in sub-saharan africa. *WORLD BANK DISCUSSION PAPERS*, 2013.

[25] Casaburi L., Kremer M., Mullainathan S., and Ramrattan R. Harnessing ict to increase agricultural production: Evidence from kenya. *Working Paper*, 2019.

[26] Samberg L.H., Gerber J.S., Ramankutty N., and Herrero M. West P.C. Subnational distribution of average farm size and smallholder contributions to global food production. *Environmental Research Letters*, 11(12):124010, 2016.

[27] Fafchamps M. and Minten B. Impact of sms-based agricultural information on indian farmers. *The World Bank Economic Review*, 26:383–414, 2012.

[28] Futch M. and McIntosh C. Tracking the introduction of the village phone product in rwanda. *Working Paper*, 2009.

[29] Gautam M. Agricultural extension : the kenya experience - an impact evaluation. *A World Bank operations evaluation study Washington, D.C. : World Bank Group*, 2000.

[30] Waskom M. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[31] Bandiera O., Burgess R., Desserano E., Morel R., Rasul I., and Sulaiman M. Development policy through the lens of social structure. *Working Paper*, 2020.

[32] ICBS (Indonesia Central Bureau of Statistics). Statistik perkebunan indonesia (tree crop estate statistics of indonesia).

[33] Courtois P. and Subervie J. Farmer bargaining power and market information services. *American Journal of Agricultural Economics*, 97:953–977, 2015.

[34] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[35] Fabregas R., Kremer M., Lowes M., On R., and Zane G. Sms-extension and farmer behavior: Lessons from six rcts in east africa. *Working Paper*, 2019.

[36] Jensen R. The digital provide: Information (technology), market performance, and welfare in the south indian fisheries sector. *The Quarterly Journal of Economics*, CXXII(3):879–924, August 2007.

[37] Tibshirani R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, 58(1):267–288, 1996.

[38] Cole S. and Fernando A.N. 'mobile'izing agricultural advice technology adoption diffusion and sustainability. *The Economic Journal*, 131:192–219, 2020.

[39] Cunningham S. *Causal Inference: The Mixtape*. Yale University Presss, 2021.

[40] Mitra S., Mookherjee D., Torero M., and Visaria S. Asymmetric information and middleman margins: An experiment with indian potato farmers. *The Review of Economics and Statistics*, 100(1):1–13, 2018.

[41] Seabold S. and Perktold J. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[42] Bruce T.J.A. The croprotect project and wider opportunities to improve farm productivity through web-based knowledge exchange. *Food and Energy Security*, 5(2):89–96, May 2016.

[43] Deichmann U., Goyal A., and Mishra D. Will digital technologies transform agriculture in developing countries? *Agricultural Economics*, 47:21–33, November 2016.

[44] Oil World.

[45] WWF. Which everyday products contain palm oil? *WWF*, 2019.

[46] Chen X., Cui Z., Fan M., Vitousek P., Zhao M., Ma W., Wang Z., Zhang W., Yan X., Yang J., Deng X., Gao Q., Zhang Q., Guo S., Ren J., Li S., Ye Y., Wang Z., Huang J., Tang ., Sun Y., Peng X., Zhang J., He M., Zhu Y., Xue J., Wang G., Wu L., An N., Wu L., Ma L., Zhang W., and Zhang F. Producing more grain with lower environmental costs. *Nature*, 514, 2014.

[47] Fu X. and Akter S. The impact of mobile phone technology on agricultural extension services delivery: Evidence from india. *The Journal of Development Studies*, 52:1561–1576, 2016.

Appendix A

Reproduction Code : Notebook 4 Price Cleaning

Notebook 4 Price Cleaning

August 29, 2021

1 Importing Library

[1]:
```
import pandas as pd
import numpy as np
import scipy
from numpy import nan
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from bokeh.plotting import figure
from bokeh.io import output_file, show, push_notebook, output_notebook
from bokeh.models import ColumnDataSource, Select
from ipywidgets import interact
import scipy.stats as st
import sys
import geopandas as gpd
import descartes
from shapely.geometry import Point, Polygon
import random
from scipy import stats
import statsmodels.api as sm
import folium
import json
import os
from IPython.display import HTML, display
import geopandas as gpd
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from numpy import array
import matplotlib.dates as mdates
from statsmodels.distributions.empirical_distribution import ECDF
import geopy.distance
from scipy.spatial import distance_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import networkx as nx
```

```
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
from sklearn.cluster import KMeans
from sklearn.neighbors import␣
↪(NeighborhoodComponentsAnalysis,KNeighborsClassifier)
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from community import community_louvain
from scipy import stats
import geopy.distance
from scipy.spatial import distance_matrix
import math
import shapely.wkt

pd.options.mode.chained_assignment = None
```

2 Importing data

[2]:
```
raw_data = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Raw Data/'

notebook_1_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 1␣
↪output/'

notebook_2_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 2␣
↪output/'

notebook_3_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 3␣
↪output/'

fig_output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 6␣
↪output/Figures/'
```

2.1 Importing users data

[3]:
```
users = pd.read_csv(f"{notebook_1_output}" + "users_processed.csv")
```

[4]:
```
users.tail()
```

[4]:

	id	name	nickname	village	village_id	photo	\
2055	2226	EKA SAPUTRA	EKA	AIR MOLEK I	331	NAN	
2056	2227	WISNU DARMAWAN	WISNU	BUKIT LINGKAR	247	NAN	
2057	2228	EDO SIBURIAN	EDO	KAMPUNG KOTA BESAR	397	NAN	
2058	2229	SUGIRAN	GIRAN	PANGKALAN KASAI	416	NAN	
2059	2230	RITA DAMEYANTI	DAME	DANAU RAMBAI	266	NAN	

```
     analysis_status analysis_note           phone  country_code  …  \
2055             NAN           NAN   +628121599004            62  …
2056             NAN           NAN  +6282211078098            62  …
2057             NAN           NAN  +6282388560874            62  …
2058             NAN           NAN  +6282319320016            62  …
2059             NAN           NAN  +6281364096543            62  …

        month  week  biweek  triweek  quadweek  year year_week year_biweek  \
2055  2021-02     5       3        2         2  2021  2021-w05   2021-bw03
2056  2021-02     5       3        2         2  2021  2021-w05   2021-bw03
2057  2021-02     5       3        2         2  2021  2021-w05   2021-bw03
2058  2021-02     6       3        2         2  2021  2021-w06   2021-bw03
2059  2021-02     6       3        2         2  2021  2021-w06   2021-bw03

     year_triweek year_quadweek
2055    2021-tw02     2021-qw02
2056    2021-tw02     2021-qw02
2057    2021-tw02     2021-qw02
2058    2021-tw02     2021-qw02
2059    2021-tw02     2021-qw02

[5 rows x 40 columns]
```

[5]:
```
users.columns
```

[5]:
```
Index(['id', 'name', 'nickname', 'village', 'village_id', 'photo',
       'analysis_status', 'analysis_note', 'phone', 'country_code',
       'phone_verified_at', 'role_id', 'sms_notification', 'push_notification',
       'status', 'src', 'referral', 'password', 'addby', 'remember_token',
       'deleted_at', 'created_at', 'updated_at', 'client_time_stamp',
       'user_id', 'village_cleaned', 'sub_district_id', 'kecamatan', 'role',
       'date', 'month', 'week', 'biweek', 'triweek', 'quadweek', 'year',
       'year_week', 'year_biweek', 'year_triweek', 'year_quadweek'],
      dtype='object')
```

[6]:
```
users['id'] = users['id'].astype(str).replace('\.0', '', regex=True)

users['village_id'] = users['village_id'].astype(str).replace('\.0', '',␣
↪regex=True)

users['user_id'] = users['user_id'].astype(str).replace('\.0', '', regex=True)

users['kecamatan_desa'] = users['kecamatan'] + '__' + users['village_cleaned']

users['kecamatan_desa'] = users['kecamatan_desa'].str.upper()
```

2.2 Importing transactions_sql data

[7]:
```
transactions_sql = (pd.read_csv(f"{notebook_1_output}"+␣
↪'transactions_sql_processed.csv', delimiter = ',')
   )
```

```
C:\Users\AK\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3156:
DtypeWarning: Columns (26,27,28) have mixed types.Specify dtype option on import
or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[8]:
```
transactions_sql['price_cleaned'].describe()
```

[8]:
```
count    31482.00000
mean      1492.04123
std        270.04120
min        500.00000
25%       1255.00000
50%       1500.00000
75%       1710.00000
max       2813.00000
Name: price_cleaned, dtype: float64
```

[9]:
```
mask = ~(transactions_sql['do_id'].isna())
transactions_sql['do_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['do_id'][mask]
                                  ]

mask = ~(transactions_sql['mill_id'].isna())
transactions_sql['mill_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['mill_id'][mask]
                                  ]

mask = ~(transactions_sql['lr_id'].isna())
transactions_sql['lr_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['lr_id'][mask]
                                  ]

mask = ~(transactions_sql['lrm_id'].isna())
transactions_sql['lrm_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['lrm_id'][mask]
                                  ]

transactions_sql['price_id'] = ['transactions_price_id_' + row for row in␣
↪transactions_sql['id'].astype(str)]

transactions_sql.tail()
```

[9]:
```
          id trans_type mill_id lr_id do_id lrm_id  user_id  grade  bridge  \
88248  89203         lr     NaN     3   NaN      3      204    NaN     NaN
88249  89204         lr     NaN     4   NaN      4      205    NaN     NaN
88250  89205         lr     NaN     4   NaN      4      205    NaN     NaN
88251  89206         lr     NaN     4   NaN      4      205    NaN     NaN
88252  89207         lr     NaN     4   NaN      4      205    NaN     NaN

            day  …  max_price_daily  price_to_mean_diff  \
88248  20210210  …           2080.0          -33.787879
88249  20210209  …           2080.0                NaN
88250  20210209  …           2080.0                NaN
88251  20210209  …           2080.0           36.212121
88252  20210209  …           2080.0           36.212121

       price_to_mean_diff_%  prev_price  prev_price_date  prev_price_diff  \
88248             -1.812861      1830.0       2021-02-09              0.0
88249                   NaN      1900.0              NaN              NaN
88250                   NaN      1900.0              NaN              NaN
88251              1.942931      1900.0       2021-02-09              0.0
88252              1.942931      1900.0       2021-02-09              0.0

       next_price  next_price_date next_price_diff  \
88248         NaN              NaN             NaN
88249         NaN              NaN             NaN
88250         NaN              NaN             NaN
88251      1900.0       2021-02-09             0.0
88252      1900.0       2021-02-09             0.0

                         price_id
88248  transactions_price_id_89203
88249  transactions_price_id_89204
88250  transactions_price_id_89205
88251  transactions_price_id_89206
88252  transactions_price_id_89207

[5 rows x 75 columns]
```

[10]:
```
transactions_sql['trans_type'].unique()
```

[10]:
```
array(['mill', 'lr'], dtype=object)
```

[11]:
```
transactions_sql['buyer_receipt'] = (transactions_sql['trans_type'] + '//' +
                                     transactions_sql['mill_id'].astype(str) +␣
↪'//' +
                                     'nan' + '//' +
                                     transactions_sql['lr_id'].astype(str)
                                    )
```

[12]:
```
transactions_sql['user_id'] = transactions_sql['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[13]:
```
transactions_sql['date'] = pd.to_datetime(transactions_sql['date']).dt.date
```

[14]:
```
transactions_sql['id']
```

[14]:
```
0           49
1           50
2           51
3           52
4           53
          …
88248    89203
88249    89204
88250    89205
88251    89206
88252    89207
Name: id, Length: 88253, dtype: int64
```

2.3 Importing last__most__transaction__do__ramp data

[15]:
```
last_most_transaction_do_ramp = pd.
↪read_csv(f"{raw_data}"+"last_most_transaction_do_ramp.csv")
```

[16]:
```
last_most_transaction_do_ramp['trans_type'] = np.nan

last_most_transaction_do_ramp['user_id'] =␣
↪last_most_transaction_do_ramp['user_id'].astype(str).replace('\.0', '',␣
↪regex=True)

mask = ~(last_most_transaction_do_ramp['do_id'].isna())
last_most_transaction_do_ramp['do_id'][mask] = [str(round(row)) for row in
                                                ␣
↪last_most_transaction_do_ramp['do_id'][mask]
                                                ]

mask = ~(last_most_transaction_do_ramp['ramp_manager_id'].isna())
last_most_transaction_do_ramp['ramp_manager_id'][mask] = [str(round(row)) for␣
↪row in
                                                          ␣
↪last_most_transaction_do_ramp['ramp_manager_id'][mask]
                                                          ]

mask = ~(last_most_transaction_do_ramp['ramp_id'].isna())
last_most_transaction_do_ramp['ramp_id'][mask] = [str(round(row)) for row in
```

```
                                                                    ␣
 ↪last_most_transaction_do_ramp['ramp_id'][mask]
                                                                    ]
last_most_transaction_do_ramp['trans_type'][mask] = 'lr'

mask = ~(last_most_transaction_do_ramp['mill_id'].isna())
last_most_transaction_do_ramp['mill_id'][mask] = [str(round(row)) for row in
                                                                    ␣
 ↪last_most_transaction_do_ramp['mill_id'][mask]
                                                                    ]
last_most_transaction_do_ramp['trans_type'][mask] = 'mill'

last_most_transaction_do_ramp['buyer'] =␣
 ↪(last_most_transaction_do_ramp['trans_type'] + '//' +
                                                                   ␣
 ↪last_most_transaction_do_ramp['mill_id'].astype(str) + '//' +
                                                                   ␣
 ↪last_most_transaction_do_ramp['do_id'].astype(str) + '//' +
                                                                   ␣
 ↪last_most_transaction_do_ramp['ramp_id'].astype(str)
                                                                   )

last_most_transaction_do_ramp['buyer_receipt'] =␣
 ↪(last_most_transaction_do_ramp['trans_type'] + '//' +
                                                                    ␣
 ↪last_most_transaction_do_ramp['mill_id'].astype(str) + '//' +
                                                                      'nan' + '//' +
                                                                    ␣
 ↪last_most_transaction_do_ramp['ramp_id'].astype(str)
                                                                    )

last_most_transaction_do_ramp.head()
```

[16]:

	id	do_name	ramp_manager_name	mill_name	ramp_name	user_id	do_id
0	1	TSR Guntur	NaN	SKIP	NaN	553	21
1	2	TSR Guntur	NaN	SKIP	NaN	553	21
2	3	TSR Guntur	NaN	SKIP	NaN	553	21
3	4	NY	NaN	Inecda	NaN	554	45
4	5	NY	NaN	Inecda	NaN	554	45

	ramp_manager_id	mill_id	ramp_id	created_at
0	NaN	1	NaN	2019-11-25 14:00:32.0
1	NaN	1	NaN	2019-11-25 14:00:32.0
2	NaN	1	NaN	2019-11-25 14:00:32.0
3	NaN	17	NaN	2019-11-27 10:56:56.0
4	NaN	17	NaN	2019-11-27 10:56:56.0

```
              updated_at      client_time_stamp             deleted_at  \
0  2019-11-25 14:00:32.0  2019-11-25 14:00:31.0                    NaN
1  2020-04-09 23:21:51.0  2019-11-25 14:00:31.0  2020-04-09 23:21:51.0
2  2020-04-09 23:21:45.0  2019-11-25 14:00:31.0  2020-04-09 23:21:45.0
3  2019-11-27 10:56:56.0  2019-11-27 10:56:54.0                    NaN
4  2020-04-09 23:21:58.0  2019-11-27 10:56:54.0  2020-04-09 23:21:58.0

  trans_type              buyer       buyer_receipt
0       mill   mill//1//21//nan   mill//1//nan//nan
1       mill   mill//1//21//nan   mill//1//nan//nan
2       mill   mill//1//21//nan   mill//1//nan//nan
3       mill  mill//17//45//nan  mill//17//nan//nan
4       mill  mill//17//45//nan  mill//17//nan//nan
```

[17]:
```
last_most_transaction_do_ramp['buyer'].unique()
```

[17]:
```
array(['mill//1//21//nan', 'mill//17//45//nan', 'mill//33//185//nan',
       'lr//nan//nan//45', 'mill//2//26//nan', 'mill//6//13//nan',
       'mill//4//50//nan', 'mill//5//nan//nan', 'mill//5//52//nan',
       'mill//6//nan//nan', 'mill//3//16//nan', 'mill//33//nan//nan',
       'lr//nan//nan//2', 'mill//32//nan//nan', 'mill//2//15//nan',
       'mill//1//nan//nan', 'mill//5//14//nan', 'mill//2//36//nan',
       'lr//nan//nan//13', 'lr//nan//nan//4', 'mill//8//nan//nan',
       'mill//2//nan//nan', 'mill//8//27//nan', 'mill//8//39//nan',
       'mill//7//46//nan', 'lr//nan//nan//43', 'mill//4//nan//nan', nan,
       'lr//nan//nan//20', 'lr//nan//nan//51', 'lr//nan//nan//22',
       'mill//42//nan//nan', 'lr//nan//nan//54', 'lr//nan//nan//26',
       'mill//13//55//nan', 'mill//13//nan//nan', 'lr//nan//nan//1',
       'mill//5//192//nan', 'mill//32//53//nan', 'mill//18//nan//nan',
       'mill//4//33//nan', 'mill//3//79//nan', 'mill//33//189//nan',
       'mill//31//nan//nan', 'mill//13//31//nan', 'lr//nan//nan//37',
       'lr//nan//nan//34', 'lr//nan//nan//33', 'lr//nan//nan//21',
       'lr//nan//nan//47', 'lr//nan//nan//38', 'mill//32//184//nan',
       'lr//nan//nan//25', 'lr//nan//nan//69', 'lr//nan//nan//50',
       'mill//7//nan//nan', 'mill//3//nan//nan', 'lr//nan//nan//66',
       'mill//76//nan//nan', 'lr//nan//nan//68', 'lr//nan//nan//49',
       'mill//47//nan//nan', 'lr//nan//nan//82', 'mill//8//191//nan',
       'mill//42//187//nan', 'lr//nan//nan//65', 'lr//nan//nan//71',
       'mill//63//nan//nan', 'mill//8//197//nan', 'lr//nan//nan//59',
       'mill//8//198//nan', 'lr//nan//nan//58', 'lr//nan//nan//70',
       'lr//nan//nan//30', 'mill//76//199//nan', 'lr//nan//nan//73',
       'lr//nan//nan//79', 'lr//nan//nan//72', 'lr//nan//nan//76',
       'mill//17//nan//nan', 'mill//49//nan//nan', 'mill//42//195//nan',
       'lr//nan//nan//85', 'lr//nan//nan//19', 'lr//nan//nan//86',
       'mill//1//21//98', 'lr//nan//nan//12', 'lr//nan//nan//89',
       'lr//nan//nan//88', 'mill//92//203//nan', 'lr//nan//nan//118',
```

```
       'mill//47//204//nan', 'lr//nan//nan//105', 'lr//nan//nan//112',
       'mill//98//205//nan', 'lr//nan//nan//120', 'lr//nan//nan//113',
       'lr//nan//nan//99', 'lr//nan//nan//114', 'mill//92//nan//nan',
       'lr//nan//nan//122', 'lr//nan//nan//92', 'lr//nan//nan//3',
       'mill//98//nan//nan', 'lr//nan//nan//115', 'mill//105//206//nan',
       'mill//80//nan//nan', 'lr//nan//nan//128', 'lr//nan//nan//24',
       'lr//nan//nan//83', 'mill//105//208//nan', 'lr//nan//nan//125',
       'lr//nan//nan//121', 'lr//nan//nan//137', 'lr//nan//nan//39',
       'lr//nan//nan//81'], dtype=object)
```

[18]:
```
last_most_transaction_do_ramp['trans_type'].value_counts()
```

[18]:
```
mill    816
lr      503
Name: trans_type, dtype: int64
```

[19]:
```
last_most_transaction_do_ramp['unknown_do'] = False

mask = ((last_most_transaction_do_ramp['trans_type']=='mill') &
        (last_most_transaction_do_ramp['do_id'].isna())
       )

last_most_transaction_do_ramp['unknown_do'][mask] = True
```

[20]:
```
mask = last_most_transaction_do_ramp['unknown_do']

len(last_most_transaction_do_ramp['user_id'][mask].unique())
```

[20]:
```
164
```

[21]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='lr') &
        (last_most_transaction_do_ramp['ramp_id'].isna())
       )

last_most_transaction_do_ramp[mask].shape
```

[21]:
```
(0, 18)
```

2.3.1 Removing the data with unknown buyer

[22]:
```
last_most_transaction_do_ramp['buyer_in_pempem'] = True
```

[23]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='mill') &
        (last_most_transaction_do_ramp['mill_id'].isna())
       )

last_most_transaction_do_ramp[mask]
```

[23]:
```
Empty DataFrame
Columns: [id, do_name, ramp_manager_name, mill_name, ramp_name, user_id, do_id,
ramp_manager_id, mill_id, ramp_id, created_at, updated_at, client_time_stamp,
deleted_at, trans_type, buyer, buyer_receipt, unknown_do, buyer_in_pempem]
Index: []
```

[24]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='mill') &
         (last_most_transaction_do_ramp['do_id'].isna())
        )

last_most_transaction_do_ramp['buyer_in_pempem'][mask] = False
```

[25]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='mill') &
         (last_most_transaction_do_ramp['do_id'].isna())
        )

last_most_transaction_do_ramp['buyer_in_pempem'][mask].value_counts()
```

[25]:
```
False    217
Name: buyer_in_pempem, dtype: int64
```

[26]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='lr') &
         (last_most_transaction_do_ramp['ramp_id'].isna())
        )

last_most_transaction_do_ramp['buyer_in_pempem'][mask] = False
```

[27]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='lr') &
         (last_most_transaction_do_ramp['ramp_id'].isna())
        )

last_most_transaction_do_ramp['buyer_in_pempem'][mask].value_counts()
```

[27]:
```
Series([], Name: buyer_in_pempem, dtype: int64)
```

[28]:
```
last_most_transaction_do_ramp['user_id'].value_counts()
```

[28]:
```
1061    7
1482    7
1575    6
761     6
1054    6
       ..
1621    1
1807    1
1712    1
1424    1
```

```
597     1
Name: user_id, Length: 949, dtype: int64
```

[29]:
```
last_most_transaction_do_ramp.shape
```

[29]:
```
(1596, 19)
```

[30]:
```
mask = last_most_transaction_do_ramp['user_id'] ==31

last_most_transaction_do_ramp[mask]
```

[30]:
```
Empty DataFrame
Columns: [id, do_name, ramp_manager_name, mill_name, ramp_name, user_id, do_id,
ramp_manager_id, mill_id, ramp_id, created_at, updated_at, client_time_stamp,
deleted_at, trans_type, buyer, buyer_receipt, unknown_do, buyer_in_pempem]
Index: []
```

[31]:
```
mask = ((last_most_transaction_do_ramp['trans_type']=='lr') &
        (last_most_transaction_do_ramp['ramp_id'].isna())
       )

last_most_transaction_do_ramp[mask]
```

[31]:
```
Empty DataFrame
Columns: [id, do_name, ramp_manager_name, mill_name, ramp_name, user_id, do_id,
ramp_manager_id, mill_id, ramp_id, created_at, updated_at, client_time_stamp,
deleted_at, trans_type, buyer, buyer_receipt, unknown_do, buyer_in_pempem]
Index: []
```

2.3.2 Merging role column to last__most__transaction__do__ramp

[32]:
```
cols = ['user_id','role']

print(last_most_transaction_do_ramp.shape)

last_most_transaction_do_ramp = last_most_transaction_do_ramp.merge(users[cols],
                                                                    on =␣
 ↪'user_id',
                                                                    how='left'
                                                                   )

print(last_most_transaction_do_ramp.shape)
```

```
(1596, 19)
(1596, 20)
```

[33]:
```
pd.pivot_table(last_most_transaction_do_ramp, index ='role',
               values='user_id',
               aggfunc=lambda x:len(x.unique()),
               margins=True
              )
```

[33]:

role	user_id
Amprah Farmer	474
Farmer	2
Middle Man	408
Ramp Manager	65
All	949

2.4 Importing user_old_buyers_survey

[34]:
```
user_old_buyers_survey = pd.read_excel(f"{notebook_3_output}" +␣
↪'user_old_buyers_df.xlsx')

user_old_buyers_survey.drop(columns= ['Unnamed: 0'], inplace=True)

user_old_buyers_survey.head()
```

[34]:

	phone	interviewee_group:name_interviewee	truck_background:mills_sale	truck_background:DOs	do_name	do_id	mill_name	user_id	role	mill_id
0	6282169650569	Beri Oktopian Sanjaya	SKIP	DO SS (Sawit sejahtera)	CV SS (TJ)	24.0	SKIP	NaN	NaN	1.0
1	6282169650569	Beri Oktopian Sanjaya	NHR	DO Irwan Simamora	NaN	NaN	NHR	NaN	NaN	2.0
2	6282390221979	Mahyudin	SRJ/NAD	SRJ DO JHON	Jhon	29.0	SRJ/NAD	147.0	Do	6.0
3	6282390221979	Mahyudin	SKIP	SKIP DO RGN	NaN	NaN	SKIP	147.0	Do	1.0
4	6282390221979	Mahyudin	NHR	NHR DO YP (Zulhendri)	YP	15.0	NHR	147.0	Do	2.0

[35]:
```
user_old_buyers_survey['user_id'] = user_old_buyers_survey['user_id'].
↪astype(str).str.extract('(\d+)', expand=False)
```

```
user_old_buyers_survey['mill_id'] = user_old_buyers_survey['mill_id'].
↪astype(str).str.extract('(\d+)', expand=False)

user_old_buyers_survey['do_id'] = user_old_buyers_survey['do_id'].astype(str).
↪str.extract('(\d+)', expand=False)
```

[36]:
```
user_old_buyers_survey.head()
```

[36]:
```
           phone interviewee_group:name_interviewee  \
0  6282169650569              Beri Oktopian Sanjaya
1  6282169650569              Beri Oktopian Sanjaya
2  6282390221979                           Mahyudin
3  6282390221979                           Mahyudin
4  6282390221979                           Mahyudin

  truck_background:mills_sale      truck_background:DOs     do_name do_id  \
0                        SKIP  DO SS ( Sawit sejahtera)  CV SS (TJ)    24
1                         NHR         DO Irwan Simamora         NaN   NaN
2                     SRJ/NAD               SRJ DO JHON        Jhon    29
3                        SKIP               SKIP DO RGN         NaN   NaN
4                         NHR    NHR DO YP ( Zulhendri)          YP    15

  mill_name user_id role mill_id
0      SKIP     NaN  NaN       1
1       NHR     NaN  NaN       2
2   SRJ/NAD     147   Do       6
3      SKIP     147   Do       1
4       NHR     147   Do       2
```

[37]:
```
user_old_buyers_survey['trans_type'] = 'mill'

user_old_buyers_survey['ramp_id'] = np.nan

#user_old_buyers_survey['do_id'] = np.nan

user_old_buyers_survey['buyer_receipt'] = (user_old_buyers_survey['trans_type']␣
↪+ '//' +
                                           user_old_buyers_survey['mill_id'].
↪astype(str) + '//' +
                                           'nan' + '//' +
                                            user_old_buyers_survey['ramp_id'].
↪astype(str)
                                            )

user_old_buyers_survey['buyer'] = (user_old_buyers_survey['trans_type'] + '//' +
                                   user_old_buyers_survey['mill_id'].
↪astype(str) + '//' +
```

```
                                    user_old_buyers_survey['do_id'].astype(str)␣
↪+ '//' +
                                    user_old_buyers_survey['ramp_id'].astype(str)
                                   )

#user_old_buyers_survey['buyer_in_pempem'] = False

#user_old_buyers_survey['unknown_do'] = True
```

[38]:
```
user_old_buyers_survey['buyer_in_pempem'] = True

mask = ((user_old_buyers_survey['trans_type']=='mill') &
         (user_old_buyers_survey['do_id'].isna())
        )

user_old_buyers_survey['buyer_in_pempem'][mask] = False
```

[39]:
```
user_old_buyers_survey['unknown_do'] = False

mask = ((user_old_buyers_survey['trans_type']=='mill') &
        (user_old_buyers_survey['do_id'].isna())
       )

user_old_buyers_survey['unknown_do'][mask] = True
```

[40]:
```
main_list = np.setdiff1d(list(last_most_transaction_do_ramp.columns),
                         list(user_old_buyers_survey.columns)
                         )

for col in main_list:
    user_old_buyers_survey[col]=np.nan
```

[41]:
```
last_most_transaction_do_ramp['source'] = 'in-app-survey'

user_old_buyers_survey['source'] = '2018 survey'
```

[42]:
```
len(last_most_transaction_do_ramp['user_id'].unique())
```

[42]: 949

[43]:
```
user_old_buyers_survey[cols]
```

[43]:

	user_id	role
0	NaN	NaN
1	NaN	NaN
2	147	Do
3	147	Do

```
4      147          Do
..     …          …
75     NaN         NaN
76      34  Middle Man
77      34  Middle Man
78      34  Middle Man
79      90  Middle Man

[80 rows x 2 columns]
```

[44]:
```
mask = user_old_buyers_survey['user_id']==str(31)

user_old_buyers_survey[mask]
```

[44]:
```
            phone interviewee_group:name_interviewee  \
36  6285376302741                                Yusup
37  6285376302741                                Yusup
38  6285376302741                                Yusup
39  6285376302741                                Yusup

   truck_background:mills_sale truck_background:DOs            do_name do_id  \
36                         SKIP     PT SKIP ( GGK )                NaN   NaN
37                         SKIP     PT SKIP ( TSR )                TSR    21
38                       Arvena     PT ARVENA ( HJ )  Harapan Jaya (HJ)    23
39                       Arvena     PT ARVENA ( SS )                NaN   NaN

   mill_name user_id        role mill_id  … unknown_do  client_time_stamp  \
36      SKIP      31  Middle Man       1  …       True                NaN
37      SKIP      31  Middle Man       1  …      False                NaN
38    Arvena      31  Middle Man       3  …      False                NaN
39    Arvena      31  Middle Man       3  …       True                NaN

   created_at deleted_at  id  ramp_manager_id  ramp_manager_name  ramp_name  \
36         NaN        NaN NaN              NaN                NaN        NaN
37         NaN        NaN NaN              NaN                NaN        NaN
38         NaN        NaN NaN              NaN                NaN        NaN
39         NaN        NaN NaN              NaN                NaN        NaN

    updated_at       source
36         NaN  2018 survey
37         NaN  2018 survey
38         NaN  2018 survey
39         NaN  2018 survey

[4 rows x 25 columns]
```

[45]:
```
cols = list(last_most_transaction_do_ramp.columns)

mask = ~(user_old_buyers_survey['user_id'].
 ↪isin(last_most_transaction_do_ramp['user_id']))

last_most_transaction_do_ramp = pd.
 ↪concat([last_most_transaction_do_ramp,user_old_buyers_survey[cols][mask]],
                                          ignore_index=True
                                         )

last_most_transaction_do_ramp.shape
```

[45]:
```
(1674, 21)
```

[46]:
```
user_old_buyers_survey['buyer'].value_counts()
```

[46]:
```
mill//1//nan//nan      19
mill//2//nan//nan      16
mill//1//24//nan       11
mill//2//15//nan        5
mill//1//21//nan        4
mill//3//16//nan        4
mill//3//nan//nan       3
mill//6//nan//nan       3
mill//3//23//nan        2
mill//2//26//nan        2
mill//nan//nan//nan     2
mill//2//36//nan        2
mill//6//13//nan        2
mill//5//nan//nan       2
mill//4//nan//nan       1
mill//6//29//nan        1
mill//8//191//nan       1
Name: buyer, dtype: int64
```

[47]:
```
mask = last_most_transaction_do_ramp['source'] == '2018 survey'
len(last_most_transaction_do_ramp['buyer_in_pempem'][mask].unique())
```

[47]:
```
2
```

[48]:
```
mask = last_most_transaction_do_ramp['user_id'] ==str(31)

last_most_transaction_do_ramp[mask]
```

[48]:
```
      id            do_name ramp_manager_name mill_name ramp_name user_id  \
1632 NaN                NaN               NaN      SKIP       NaN      31
1633 NaN                TSR               NaN      SKIP       NaN      31
```

```
1634 NaN  Harapan Jaya (HJ)               NaN    Arvena       NaN      31
1635 NaN                NaN               NaN    Arvena       NaN      31

     do_id ramp_manager_id mill_id ramp_id  … updated_at client_time_stamp  \
1632   NaN             NaN       1     NaN  …        NaN               NaN
1633    21             NaN       1     NaN  …        NaN               NaN
1634    23             NaN       3     NaN  …        NaN               NaN
1635   NaN             NaN       3     NaN  …        NaN               NaN

     deleted_at trans_type              buyer      buyer_receipt unknown_do  \
1632        NaN       mill  mill//1//nan//nan  mill//1//nan//nan       True
1633        NaN       mill   mill//1//21//nan  mill//1//nan//nan      False
1634        NaN       mill   mill//3//23//nan  mill//3//nan//nan      False
1635        NaN       mill  mill//3//nan//nan  mill//3//nan//nan       True

      buyer_in_pempem        role       source
1632            False  Middle Man  2018 survey
1633             True  Middle Man  2018 survey
1634             True  Middle Man  2018 survey
1635            False  Middle Man  2018 survey

[4 rows x 21 columns]
```

2.5 Importing do_ramp_visit data

[49]:
```
do_ramp_visit = pd.read_csv(f"{notebook_1_output}"+'do_ramp_visit_processed.
↪csv')

do_ramp_visit['date_time'] = pd.to_datetime(do_ramp_visit['date_time'])

do_ramp_visit['date'] = pd.to_datetime(do_ramp_visit['date']).dt.date

do_ramp_visit.head()
```

```
C:\Users\AK\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3156:
DtypeWarning: Columns (16) have mixed types.Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[49]:
```
   id  user_id  do_id           date_time        date    month  week  biweek  \
0   3      124    NaN 2019-05-09 03:28:18  2019-05-09  2019-05    19      10
1   5      201    NaN 2019-05-09 06:27:29  2019-05-09  2019-05    19      10
2   6      201    NaN 2019-05-09 06:27:33  2019-05-09  2019-05    19      10
3   7      201    NaN 2019-05-09 06:27:35  2019-05-09  2019-05    19      10
4   8      201    NaN 2019-05-09 06:27:39  2019-05-09  2019-05    19      10

   triweek  quadweek  …         name          phone        role mill_id  \
```

```
0        7         5  …  MOHAMAD ALI    62818496476  Middle Man     NaN
1        7         5  …         SIMON  6285248116660  Middle Man     NaN
2        7         5  …         SIMON  6285248116660  Middle Man     NaN
3        7         5  …         SIMON  6285248116660  Middle Man     NaN
4        7         5  …         SIMON  6285248116660  Middle Man     NaN

  price_sharing Sharing or Viewing trans_type lr_id             buyer  \
0           NaN       Viewing Only         lr  11.0  lr//nan//nan//11
1           NaN       Viewing Only         lr   3.0   lr//nan//nan//3
2           NaN       Viewing Only         lr   5.0   lr//nan//nan//5
3           NaN       Viewing Only         lr   6.0   lr//nan//nan//6
4           NaN       Viewing Only         lr  12.0  lr//nan//nan//12

      buyer_receipt
0  lr//nan//nan//11
1   lr//nan//nan//3
2   lr//nan//nan//5
3   lr//nan//nan//6
4  lr//nan//nan//12

[5 rows x 25 columns]
```

[50]:
```
do_ramp_visit.columns
```

[50]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt'],
      dtype='object')
```

[51]:
```
do_ramp_visit['id'] = do_ramp_visit['id'].astype(str).replace('\.0', '',␣
↪regex=True)

do_ramp_visit['user_id'] = do_ramp_visit['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)

do_ramp_visit['do_id'] = do_ramp_visit['do_id'].astype(str).replace('\.0', '',␣
↪regex=True)

do_ramp_visit['mill_id'] = do_ramp_visit['mill_id'].astype(str).replace('\.0',␣
↪'', regex=True)

do_ramp_visit['lr_id'] = do_ramp_visit['lr_id'].astype(str).replace('\.0', '',␣
↪regex=True)
```

[52]:
```
do_ramp_visit['trans_type'] = np.nan
```

[53]:
```
mask = ~(do_ramp_visit['lr_id'].isna())

do_ramp_visit['trans_type'][mask] = 'lr'
```

[54]:
```
mask = ~(do_ramp_visit['mill_id'].isna())

do_ramp_visit['trans_type'][mask] = 'mill'
```

[55]:
```
do_ramp_visit['trans_type'].unique()
```

[55]:
```
array(['mill'], dtype=object)
```

2.6 Importing login_activities data

[56]:
```
login_activities = pd.read_csv(f"{raw_data}"+'login_activities.csv')

login_activities.head()
```

[56]:

	id	user_id	user_agent	ip_address	app_version	created_at	updated_at	lng	lat	deleted_at
0	9	25	7.1.1	114.125.57.152	1.7	2019-01-08 08:39:07.0	2019-01-08 08:39:07.0	NaN	NaN	NaN
1	10	25	7.1.1	114.125.57.152	1.7	2019-01-08 08:39:42.0	2019-01-08 08:39:42.0	NaN	NaN	NaN
2	11	36	7.0	114.125.52.152	1.7	2019-01-08 09:20:23.0	2019-01-08 09:20:23.0	NaN	NaN	NaN
3	12	36	7.0	114.125.52.152	1.7	2019-01-08 09:22:22.0	2019-01-08 09:22:22.0	NaN	NaN	NaN
4	13	81	8.0.0	114.125.40.156	1.7	2019-01-08 09:24:12.0	2019-01-08 09:24:12.0	NaN	NaN	NaN

[57]:
```
login_activities['long_rounded'] = round(login_activities['lng'],3)
login_activities['lat_rounded'] = round(login_activities['lat'],3)
```

2.7 Importing do data

[58]:
```
do = pd.read_csv(f"{raw_data}"+"do.csv"
                 )


do.head()
```

[58]:

	id	user_id	name	lng	lat	address
0	13	23	MR 89	102.501629	-0.682737	Desa Sungai Akar
1	14	24	ABP	102.654415	-0.712050	lintas samudra km 9
2	15	25	YP	102.524086	-0.734713	Seberida

```
3  16     1152     AG  102.345258 -0.662747  CV. ALFA GLOBAL DESA PEJANGKI
4  18       42    TSR  102.411121 -0.558998     jalan Lintas Timur Belilas

           description         phone  q1  q2  …   c1   c2  c3  addby  \
0        Jalan PT. NAD  6.282287e+12   0   0  …  NaN  NaN NaN    api
1      Desa petalongan  6.281221e+11   0   0  …  NaN  NaN NaN    api
2  Jalan Masuk PKS NHR  6.282382e+12   0   0  …  NaN  NaN NaN    api
3   Kantin Alfa Global  6.282387e+12   0   0  …  NaN  NaN NaN    api
4                  NaN  6.281271e+12   0   0  …  NaN  NaN NaN    api

  cash_visibility  status  client_time_stamp            deleted_at  \
0               1       1                NaN                   NaN
1               1       1                NaN                   NaN
2               1       1                NaN                   NaN
3               1       1                NaN                   NaN
4               1       2                NaN  2019-11-04 09:56:47.0

              created_at            updated_at
0  2018-12-01 13:46:42.0  2019-09-04 10:01:15.0
1  2018-12-02 09:54:00.0  2019-06-11 16:48:51.0
2  2018-12-02 17:14:12.0  2019-08-01 00:42:46.0
3  2018-12-03 14:22:09.0  2020-06-23 13:31:49.0
4  2018-12-05 15:24:54.0  2019-11-04 09:56:47.0

[5 rows x 21 columns]
```

[59]:
```
do['do_id']=do['id'].astype(str).replace('\.0', '', regex=True)
do['id']=do['id'].astype(str).replace('\.0', '', regex=True)
```

[60]:
```
len(last_most_transaction_do_ramp['do_id'].unique())
```

[60]:
```
37
```

2.7.1 Merging Buyer column

[61]:
```
do.columns
```

[61]:
```
Index(['id', 'user_id', 'name', 'lng', 'lat', 'address', 'description',
       'phone', 'q1', 'q2', 'q3', 'c1', 'c2', 'c3', 'addby', 'cash_visibility',
       'status', 'client_time_stamp', 'deleted_at', 'created_at', 'updated_at',
       'do_id'],
      dtype='object')
```

[62]:
```
cols = ['do_id', 'buyer','buyer_receipt']

print(do.shape)
```

```
do = do.merge(do_ramp_visit[cols].drop_duplicates(),
              on='do_id',
              how='left'
             )

print(do.shape)
```

```
(50, 22)
(50, 24)
```

2.8 Importing lr_lrm data

[63]:
```
lr_lrm = pd.read_csv(f"{raw_data}"+"lr_lrm.csv")

lr_lrm.head()
```

[63]:

	id	lrm_id	lr_id	client_time_stamp	deleted_at	created_at	updated_at
0	1	1	1	NaN	NaN	2019-04-11 09:27:39.0	2019-04-11 09:27:39.0
1	2	2	2	NaN	NaN	2019-04-11 11:08:17.0	2019-04-11 11:08:17.0
2	3	3	3	NaN	NaN	2019-04-11 12:19:46.0	2019-04-11 12:19:46.0
3	4	4	4	NaN	NaN	2019-04-11 14:15:16.0	2019-04-11 14:15:16.0
4	5	5	5	NaN	NaN	2019-05-08 10:25:39.0	2019-05-08 10:25:39.0

[64]:
```
lr_lrm['id']=lr_lrm['id'].astype(str).replace('\.0', '', regex=True)

lr_lrm['lrm_id']=lr_lrm['lrm_id'].astype(str).replace('\.0', '', regex=True)

lr_lrm['lr_id']=lr_lrm['lr_id'].astype(str).replace('\.0', '', regex=True)
```

2.9 Importing lrm data

[65]:
```
lrm = pd.read_csv(f"{raw_data}"+"lrm.csv")

lrm.head()
```

[65]:

	id	user_id	name	lng	lat	address
0	1	202	Juan Purba	102.668298	-0.889027	Dusun Leboy
1	2	203	Joni	102.514567	-0.734371	Desa Seberida
2	3	204	Mastuni	102.423992	-0.628439	Pangkalan Kasai
3	4	205	Ponijan	102.412089	-0.551158	Seresam

```
4   5      218    Pak Joko  102.257117 -0.658824      Bukit Lipai

                            description  q1  q2  q3   c1   c2   c3 addby  \
0  Masad Lima, Depan Warung Makan Sagala   0   0   0  NaN  NaN  NaN   api
1         Depan Rumah Makan Segar Mangan   0   0   0  NaN  NaN  NaN   api
2                                Pegegas   0   0   0  NaN  NaN  NaN   api
3                                 Blok E   0   0   0  NaN  NaN  NaN   api
4                  Sebelah cucian mobil   0   0   0  NaN  NaN  NaN   api

   cash_visibility  status client_time_stamp deleted_at  \
0                1       2               NaN        NaN
1                1       1               NaN        NaN
2                1       1               NaN        NaN
3                1       1               NaN        NaN
4                1       2               NaN        NaN

              created_at             updated_at
0  2019-04-11 09:27:39.0  2020-06-12 10:46:23.0
1  2019-04-11 11:08:17.0  2019-11-09 04:54:39.0
2  2019-04-11 12:19:46.0  2019-08-01 00:42:47.0
3  2019-04-11 14:15:16.0  2019-08-01 00:42:47.0
4  2019-05-08 10:25:39.0  2020-09-10 05:28:22.0
```

[66]:
```
lrm.shape
```

[66]:
```
(101, 20)
```

[67]:
```
lrm['id'] = lrm['id'].astype(str).replace('\.0', '', regex=True)

lrm['lrm_id'] = lrm['id'].astype(str).replace('\.0', '', regex=True)

lrm['user_id'] = lrm['user_id'].astype(str).replace('\.0', '', regex=True)
```

2.9.1 Merging lrm_id column

[68]:
```
lrm.columns
```

[68]:
```
Index(['id', 'user_id', 'name', 'lng', 'lat', 'address', 'description', 'q1',
       'q2', 'q3', 'c1', 'c2', 'c3', 'addby', 'cash_visibility', 'status',
       'client_time_stamp', 'deleted_at', 'created_at', 'updated_at',
       'lrm_id'],
      dtype='object')
```

[69]:
```
cols = ['lrm_id','lr_id']

print(lrm.shape)
```

```
lrm = lrm.merge(lr_lrm[cols],
                on = 'lrm_id',
                how='left'
               )

print(lrm.shape)
```

```
(101, 21)
(101, 22)
```

2.9.2 Merging Buyer column

[70]:
```
do_ramp_visit.columns
```

[70]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt'],
      dtype='object')
```

[71]:
```
cols = ['lr_id', 'buyer','buyer_receipt']

print(lrm.shape)

lrm = lrm.merge(do_ramp_visit[cols].drop_duplicates(),
                on='lr_id',
                how='left'
               )

print(lrm.shape)
```

```
(101, 22)
(101, 24)
```

2.10 Importing relevant_village

[72]:
```
relevant_village = pd.
 ↪read_excel(f"{notebook_2_output}"+"relevant_village_complete.xlsx")

geom_list = []

for geom in relevant_village['geometry']:
    try:
        geom_list.append(shapely.wkt.loads(geom))
    except:
        geom_list.append(np.nan)
```

```
relevant_village['geometry'] = geom_list

mask = relevant_village['geometry'].notna()

relevant_village = relevant_village[mask]

#relevant_village.reset_index(inplace = True)

relevant_village.drop(columns=['Unnamed: 0','index'],inplace=True)

relevant_village.head()
```

```
ParseException: Expected word but encountered end of stream
```

[72]:

	level_0	PROVNO	KABKOTNO	KECNO	DESANO	IDSP2010	PROVINSI	KABKOT	\
0	0	14	4	20	30	1404020030	RIAU	PELALAWAN	
1	1	14	4	21	10	1404021010	RIAU	PELALAWAN	
2	2	14	4	32	11	1404032011	RIAU	PELALAWAN	
3	3	14	4	32	6	1404032006	RIAU	PELALAWAN	
4	4	14	4	30	13	1404030013	RIAU	PELALAWAN	

	KECAMATAN	DESA	SUMBER	\
0	PANGKALAN KURAS	PALAS	SP2010	
1	UKUI	AIR EMAS	SP2010	
2	BANDAR PETALANGAN	AIR TERJUN	SP2010	
3	BANDAR PETALANGAN	ANGKASA	SP2010	
4	BUNUT	BAGAN LAGUH	SP2010	

	geometry	\
0	(POLYGON ((101.9055505690001 0.275409893000073…	
1	POLYGON ((102.1206216090001 -0.109291733999953…	
2	POLYGON ((102.1467148460001 0.1343687620000651…	
3	POLYGON ((102.125160625 0.189884060000054, 102…	
4	POLYGON ((102.0921405430001 0.2523691220000615…	

	kecamatan_desa	relevant_village_index	nighttime_light	\
0	PANGKALAN KURAS__PALAS	0.0	65.637956	
1	UKUI__AIR EMAS	1.0	43.561230	
2	BANDAR PETALANGAN__AIR TERJUN	2.0	39.424383	
3	BANDAR PETALANGAN__ANGKASA	3.0	15.253111	
4	BUNUT__BAGAN LAGUH	4.0	141.456076	

	area	nighttime_light_density
0	38.116357	1.722042
1	16.328344	2.667829
2	34.678845	1.136842

```
3   22.252859                 0.685445
4  127.280617                 1.111372
```

[73]:
```
connectedness_village_df = pd.
↪read_excel(f"{notebook_2_output}"+"connectedness_village_df.xlsx")

connectedness_village_df.drop(columns=['Unnamed: 0'],inplace=True)

connectedness_village_df.head()
```

[73]:
```
            buyer_kecamatan_desa          kecamatan_desa  \
0         PANGKALAN KURAS__PALAS  PANGKALAN KURAS__PALAS
1                UKUI__AIR EMAS  PANGKALAN KURAS__PALAS
2  BANDAR PETALANGAN__AIR TERJUN  PANGKALAN KURAS__PALAS
3     BANDAR PETALANGAN__ANGKASA  PANGKALAN KURAS__PALAS
4             BUNUT__BAGAN LAGUH  PANGKALAN KURAS__PALAS

   kecamatan_desa_connected  same_village     user_kecamatan_desa  \
0                         0          True  PANGKALAN KURAS__PALAS
1                         0         False  PANGKALAN KURAS__PALAS
2                         0         False  PANGKALAN KURAS__PALAS
3                         0         False  PANGKALAN KURAS__PALAS
4                         1         False  PANGKALAN KURAS__PALAS

   shortest_path_length
0                   0.0
1                   6.0
2                   5.0
3                   3.0
4                   1.0
```

[74]:
```
village_neighbors_df = pd.
↪read_excel(f"{notebook_2_output}"+"village_neighbors_df.xlsx")

village_neighbors_df.drop(columns=['Unnamed: 0'],inplace=True)

village_neighbors_df.head()
```

[74]:
```
                kecamatan_desa           user_kecamatan_desa  \
0         PANGKALAN KURAS__PALAS         PANGKALAN KURAS__PALAS
1                UKUI__AIR EMAS                UKUI__AIR EMAS
2  BANDAR PETALANGAN__AIR TERJUN  BANDAR PETALANGAN__AIR TERJUN
3     BANDAR PETALANGAN__ANGKASA     BANDAR PETALANGAN__ANGKASA
4             BUNUT__BAGAN LAGUH             BUNUT__BAGAN LAGUH

                              village_neighbors_<=5  \
0  ['PANGKALAN KURAS__PALAS', 'BANDAR PETALANGAN_…
```

```
1  ['UKUI__AIR EMAS', 'BANDAR PETALANGAN__AIR TER…
2  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
3  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
4  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…

                                village_neighbors_<=8
0  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
1  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
2  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
3  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
4  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
```

2.11 Importing price__group__user data

[75]:
```
price_group_user = pd.read_csv(f"{raw_data}"+'price_group_user.csv')

price_group_user['date'] = pd.
 ↪to_datetime(price_group_user['client_time_stamp']).dt.date

price_group_user['date_added'] = price_group_user['date']

price_group_user.tail()
```

[75]:
```
     price_group_user_id  price_group_id  user_id      client_time_stamp  \
758                  759             101     2194  2021-01-27 21:57:53.0
759                  760             101     2195  2021-01-27 21:57:53.0
760                  761             101     2196  2021-01-27 21:57:53.0
761                  762             101     1702  2021-01-27 21:57:53.0
762                  763             101     2197  2021-01-27 21:57:53.0

                created_at              updated_at deleted_at        date  \
758  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
759  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
760  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
761  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
762  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27

     date_added
758  2021-01-27
759  2021-01-27
760  2021-01-27
761  2021-01-27
762  2021-01-27
```

[76]:
```
price_group_user.columns
```

[76]:
```
Index(['price_group_user_id', 'price_group_id', 'user_id', 'client_time_stamp',
       'created_at', 'updated_at', 'deleted_at', 'date', 'date_added'],
      dtype='object')
```

[77]:
```
price_group_user['price_group_id'] = price_group_user['price_group_id'].
↪astype(str).replace('\.0', '', regex=True)

price_group_user['price_group_user_id'] =␣
↪price_group_user['price_group_user_id'].astype(str).replace('\.0', '',␣
↪regex=True)

price_group_user['user_id'] = price_group_user['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[78]:
```
price_group = pd.read_csv(f"{raw_data}"+'price_group.csv',sep=',')

price_group['date'] = pd.to_datetime(price_group['client_time_stamp']).dt.date

price_group.tail()
```

[78]:

	price_group_id	name	added_by_user_id	identifier	client_time_stamp	created_at	updated_at	deleted_at	date
96	97	PTN 062	1569	8a85e495-0792-4709-b120-bd6fd83208f4	2020-10-25 10:06:23.0	2020-10-25 10:06:26.0	2020-10-25 10:06:26.0	NaN	2020-10-25
97	98	Hasri	1596	9b831549-6d5b-44d0-83c5-55dd76a496d1	2020-11-02 09:46:52.0	2020-11-02 09:46:54.0	2020-11-20 16:12:02.0	NaN	2020-11-02
98	99	RAMP FORESTER	1432	191a5cb3-1b40-4f61-9507-98cf87a6b5cd	2020-11-09 16:49:00.0	2020-11-09 16:48:30.0	2020-11-09 16:49:03.0	NaN	2020-11-09
99	100	group A	1600	88040afb-1d8d-479c-9be2-248bc3255872	2020-11-19 11:32:47.0	2020-11-19 11:32:50.0	2020-11-20 07:34:23.0	NaN	2020-11-19
100	101	Ramp putra tunggal simpang wks	1701	b8664ac6-90d8-472b-ae82-823fcedbd942	2021-01-27 21:57:53.0	2021-01-27 21:57:54.0	2021-01-29 14:47:05.0	NaN	2021-01-27

[79]:
```
price_group['price_group_id'] = price_group['price_group_id'].astype(str).
↪replace('\.0', '', regex=True)

price_group['added_by_user_id'] = price_group['added_by_user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[80]:
```
price_group_user.tail()
```

[80]:
```
     price_group_user_id price_group_id user_id      client_time_stamp  \
758                  759            101    2194  2021-01-27 21:57:53.0
759                  760            101    2195  2021-01-27 21:57:53.0
760                  761            101    2196  2021-01-27 21:57:53.0
761                  762            101    1702  2021-01-27 21:57:53.0
762                  763            101    2197  2021-01-27 21:57:53.0

                created_at             updated_at deleted_at        date  \
758  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
759  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
760  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
761  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27
762  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0        NaN  2021-01-27

     date_added
758  2021-01-27
759  2021-01-27
760  2021-01-27
761  2021-01-27
762  2021-01-27
```

[81]:
```
price_group_price = pd.read_csv(f"{raw_data}"+'price_group_price.csv',sep=',')

price_group_price['date'] = [str(row)[:4]+'/'+str(row)[4:6]+'/'+str(row)[-2:]
                             for row in price_group_price['day']]

price_group_price['date'] = pd.to_datetime(price_group_price['date']).dt.date

price_group_price.tail()
```

[81]:
```
      price_group_price_id  price_group_id  added_by_user_id  \
2707                  2708              31             495.0
2708                  2709              29             239.0
2709                  2710              30             239.0
2710                  2711              47             470.0
2711                  2712              48             470.0

      added_by_backend_user_id       day  price  published  \
2707                       NaN  20210210   1900          1
2708                       NaN  20210210   1800          1
2709                       NaN  20210210   1800          1
2710                       NaN  20210210   1900          1
2711                       NaN  20210210   1890          1

          client_time_stamp             created_at             updated_at  \
```

```
2707  2021-02-09 19:29:45.0  2021-02-09 19:30:06.0  2021-02-10 00:00:01.0
2708  2021-02-09 20:54:26.0  2021-02-09 20:54:33.0  2021-02-10 00:00:01.0
2709  2021-02-09 20:54:20.0  2021-02-09 20:54:33.0  2021-02-10 00:00:01.0
2710  2021-02-09 22:08:42.0  2021-02-09 22:08:44.0  2021-02-10 00:00:01.0
2711  2021-02-09 22:08:42.0  2021-02-09 22:08:44.0  2021-02-10 00:00:01.0
```

```
     deleted_at        date
2707        NaN  2021-02-10
2708        NaN  2021-02-10
2709        NaN  2021-02-10
2710        NaN  2021-02-10
2711        NaN  2021-02-10
```

[82]:
```
price_group_price['price_group_price_id'] =␣
↪price_group_price['price_group_price_id'].astype(str).replace('\.0', '',␣
↪regex=True)

price_group_price['price_group_id'] = price_group_price['price_group_id'].
↪astype(str).replace('\.0', '', regex=True)

price_group_price['added_by_user_id'] = price_group_price['added_by_user_id'].
↪astype(str).replace('\.0', '', regex=True)
```

[83]:
```
print(price_group_user.shape)

cols = ['user_id','role']

price_group_user = price_group_user.merge(users[cols],
                                          on='user_id',
                                          how= 'left'
                                         )


print(price_group_user.shape)
```

```
(763, 9)
(763, 10)
```

[84]:
```
print(price_group.shape)

cols = ['user_id','role']

price_group = price_group.merge(users[cols],
                                left_on='added_by_user_id',
                                right_on='user_id',
                                how= 'left'
                                )
```

```
print(price_group.shape)
```

```
(101, 9)
(101, 11)
```

[85]:
```
price_group['role'].value_counts()
```

[85]:
```
Ramp Manager    71
Do              27
Farmer           2
Middle Man       1
Name: role, dtype: int64
```

[86]:
```
price_group_user['role'].value_counts()
```

[86]:
```
Middle Man       625
Amprah Farmer    129
Ramp Manager       5
Farmer             2
Do                 1
Do/Middle Man      1
Name: role, dtype: int64
```

[87]:
```
price_group['role'].value_counts()
```

[87]:
```
Ramp Manager    71
Do              27
Farmer           2
Middle Man       1
Name: role, dtype: int64
```

[88]:
```
print(price_group_user.shape)

cols = ['added_by_user_id','price_group_id']

price_group_user = price_group_user.merge(price_group[cols],
                                          on='price_group_id',
                                          how= 'left'
                                         )

print(price_group_user.shape)
```

```
(763, 10)
(763, 11)
```

2.12 Importing fruit sale table

[89]:
```
fruit_sale = pd.read_csv(f"{raw_data}"+ "fruit_sale.csv")

fruit_sale['fruit_sale_id'] = fruit_sale['fruit_sale_id'].astype(str).
 ↪replace('\.0', '', regex=True)

fruit_sale['date'] = pd.to_datetime(fruit_sale['created_at']).dt.date

fruit_sale['receipt_id'] = ['fs_' + str(row) for row in␣
 ↪fruit_sale['fruit_sale_id']]

fruit_sale['price_id'] = ['fs_price_id_' + row for row in␣
 ↪fruit_sale['fruit_sale_id'].astype(str)]

fruit_sale.head()
```

[89]:

	fruit_sale_id	fruit_sale_identifier	telemetry_log_id	\
0	598	33988ff6-bf17-4a3a-a9d3-38328100142f	NaN	
1	599	779c2c75-6269-4c54-8a3d-bcb0fcb430b5	NaN	
2	601	c61bc660-1d5e-4992-91e2-8e4d4e8ddcc3	NaN	
3	602	224a39c6-2bee-4c5a-8105-64db0314d6e7	NaN	
4	604	0c137e5a-a956-43a5-a74b-0a7b55dd2846	NaN	

	buyer_type	user_id	total_weight	total_sale	payment_date	\
0	NaN	1585	1500.0	1500000.0	2020-10-30 00:00:00.0	
1	NaN	1583	1160.0	1650000.0	2020-10-30 00:00:00.0	
2	do	1059	1800.0	NaN	NaN	
3	do	97	2000.0	NaN	NaN	
4	mill	1062	7000.0	NaN	NaN	

	adjusted_weight	bridge	…	is_shared	created_client_time_stamp	\
0	NaN	NaN	…	0.0	NaN	
1	NaN	NaN	…	0.0	NaN	
2	80.0	1.0	…	0.0	2020-10-30 10:39:39.0	
3	120.0	2.0	…	0.0	2020-10-30 10:40:42.0	
4	350.0	1.0	…	0.0	2020-10-30 10:41:09.0	

	app_version	updated_client_time_stamp	created_at	\
0	NaN	2020-10-30 09:57:49.0	2020-10-30 09:57:52.0	
1	NaN	2020-10-30 09:59:56.0	2020-10-30 09:59:59.0	
2	2.7.0	2020-10-30 10:40:59.0	2020-10-30 10:39:41.0	
3	2.7.0	2020-10-30 10:40:42.0	2020-10-30 10:40:43.0	
4	2.7.0	2020-10-30 10:41:09.0	2020-10-30 10:41:11.0	

	updated_at	deleted_at	date	receipt_id	\
0	2020-10-30 09:57:52.0	NaN	2020-10-30	fs_598	

```
1  2020-10-30 09:59:59.0                   NaN  2020-10-30     fs_599
2  2020-11-17 04:38:05.0  2020-11-17 04:38:05.0  2020-10-30     fs_601
3  2020-12-01 05:42:10.0  2020-12-01 05:42:10.0  2020-10-30     fs_602
4  2020-12-01 05:41:45.0  2020-12-01 05:41:45.0  2020-10-30     fs_604

          price_id
0  fs_price_id_598
1  fs_price_id_599
2  fs_price_id_601
3  fs_price_id_602
4  fs_price_id_604

[5 rows x 22 columns]
```

[90]:
```
fruit_sale_transformed_transaction = pd.read_csv(f"{raw_data}"+␣
↪"fruit_sale_transformed_transaction.csv")

fruit_sale_transformed_transaction['fruit_sale_id'] =␣
↪fruit_sale_transformed_transaction['fruit_sale_id'].astype(str).replace('\.
↪0', '', regex=True)

fruit_sale_transformed_transaction['transaction_id'] =␣
↪fruit_sale_transformed_transaction['transaction_id'].astype(str).replace('\.
↪0', '', regex=True)

fruit_sale_transformed_transaction['date'] = pd.
↪to_datetime(fruit_sale_transformed_transaction['created_at']).dt.date

fruit_sale_transformed_transaction.head()
```

[90]:
```
   transaction_id  user_id trans_type  mill_id  do_id  ramp_id  \
0            2272      201       mill      2.0   26.0      NaN
1            2273      201       mill      2.0   26.0      NaN
2            2274      201       mill      2.0   26.0      NaN
3            2275      201       mill      2.0   26.0      NaN
4            2276      201       mill      2.0   26.0      NaN

   ramp_manager_id other_do  grade  bridge  …  total_weight  \
0              NaN      NaN    0.0     NaN  …           NaN
1              NaN      NaN    0.0     NaN  …           NaN
2              NaN      NaN    0.0     NaN  …           NaN
3              NaN      NaN    0.0     NaN  …           NaN
4              NaN      NaN    0.0     NaN  …           NaN

   adjusted_weight      client_time_stamp  app_version  fruit_sale_id  \
0              NaN  2020-11-02 03:00:31.0        2.7.0           3061
1              NaN  2020-11-02 03:00:31.0        2.7.0           3061
```

```
2              NaN  2020-11-02 03:00:31.0        2.7.0           3061
3              NaN  2020-11-02 03:00:31.0        2.7.0           3061
4              NaN  2020-11-02 03:00:31.0        2.7.0           3061

   transaction_split_type             deleted_at             created_at  \
0                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
1                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
2                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
3                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
4                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0

              updated_at        date
0  2020-11-02 04:00:31.0  2020-11-02
1  2020-11-02 04:00:31.0  2020-11-02
2  2020-11-02 04:00:31.0  2020-11-02
3  2020-11-02 04:00:31.0  2020-11-02
4  2020-11-02 04:00:31.0  2020-11-02

[5 rows x 28 columns]
```

[91]:
```
fruit_sale['price'].describe()
```

[91]:
```
count    4.326000e+03
mean     2.166486e+03
std      1.530332e+04
min      1.700000e+01
25%      1.850000e+03
50%      1.900000e+03
75%      1.960000e+03
max      1.001000e+06
Name: price, dtype: float64
```

[92]:
```
fruit_sale['price'].dropna().sort_values().tail(20)
```

[92]:
```
4041       2100.0
4496       2100.0
4500       2120.0
4156       2120.0
4267       2120.0
4234       2120.0
4036       2200.0
4081       2200.0
3319       2750.0
3300       2800.0
2557       2990.0
2718       3240.0
4310       3333.0
```

```
1006       4690.0
3133       4860.0
735        6150.0
868        7040.0
940        9130.0
4606     123654.0
5240    1001000.0
Name: price, dtype: float64
```

[93]:
```
mask = fruit_sale['price'] >2200

fruit_sale['price'][mask] = np.nan
```

[94]:
```
mask = fruit_sale['total_weight'] >20000

fruit_sale['total_weight'][mask] = np.nan
```

[95]:
```
mask = fruit_sale['adjusted_weight'] >3000

fruit_sale['adjusted_weight'][mask] = np.nan
```

[96]:
```
print(fruit_sale.shape)

cols =␣
 ↪['fruit_sale_id','trans_type','mill_id','ramp_id','do_id','ramp_manager_id']

fruit_sale = fruit_sale.merge(fruit_sale_transformed_transaction[cols].
 ↪drop_duplicates(),
                              on = 'fruit_sale_id',
                              how = 'left'
                             )

print(fruit_sale.shape)
```

```
(5619, 22)
(5619, 27)
```

[97]:
```
print(fruit_sale.shape)

cols = ['fruit_sale_id','grade']

fruit_sale = fruit_sale.merge(fruit_sale_transformed_transaction[cols].dropna().
 ↪drop_duplicates(),
                              on = 'fruit_sale_id',
                              how = 'left'
                             )
```

```
print(fruit_sale.shape)
```

```
(5619, 27)
(5619, 28)
```

[98]:
```
mask = fruit_sale['grade'] >15

fruit_sale['grade'][mask] = np.nan
```

[99]:
```
mask = fruit_sale['grade'] <1

fruit_sale['grade'][mask] = np.nan
```

[100]:
```
fruit_sale['mill_id'] = fruit_sale['mill_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['lr_id'] = fruit_sale['ramp_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['do_id'] = fruit_sale['do_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['lrm_id'] = fruit_sale['ramp_manager_id'].astype(str).replace('\.0',␣
↪'', regex=True)

fruit_sale['user_id'] = fruit_sale['user_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale.replace('nan',np.nan, inplace=True)

fruit_sale['buyer_receipt'] = (fruit_sale['trans_type'] + '//' +
                               fruit_sale['mill_id'].astype(str) + '//' +
                               'nan' + '//' +
                                fruit_sale['lr_id'].astype(str)
                              )

fruit_sale['buyer'] = (fruit_sale['trans_type'] + '//' +
                       fruit_sale['mill_id'].astype(str) + '//' +
                       fruit_sale['do_id'].astype(str) + '//' +
                       fruit_sale['lr_id'].astype(str)
                      )

fruit_sale.rename(columns={'grade':'grade_fruit_sale',
                           'price':'price_fruit_sale'
                          }, inplace=True)
```

[101]:
```
fruit_sale['grade_fruit_sale'].describe()
```

[101]:
```
count    3429.000000
mean        4.442724
std         1.058820
min         1.000000
25%         3.900000
50%         4.490000
75%         5.000000
max        14.810000
Name: grade_fruit_sale, dtype: float64
```

[102]:
```
mask = fruit_sale['price_fruit_sale'] ==1000

fruit_sale[['date','buyer']][mask]
```

[102]:

	date	buyer
9	2020-10-30	NaN
10	2020-10-30	NaN
16	2020-10-30	mill//2//26//nan
3327	2020-12-21	lr//nan//nan//57
3398	2020-12-23	mill//8//nan//nan
4125	2021-01-12	mill//17//45//nan
4777	2021-01-23	mill//2//nan//nan

2.13 Importing manual price imputation table

[103]:
```
users_receipt_duplicates = pd.
↪read_excel(f"{raw_data}"+"users_receipt_duplicates_20210303.xlsx",
                                        #sep=";"
                                       )



users_receipt_duplicates['date'] = pd.
↪to_datetime(users_receipt_duplicates['date']).dt.date

users_receipt_duplicates['imputed_post_grading_weight'].replace({0:np.
↪nan},inplace =True)

users_receipt_duplicates['imputed_price_manual'].replace({0:np.nan},inplace␣
↪=True)

users_receipt_duplicates['price_id'] = ['manual_price_id_' + row for row in␣
↪users_receipt_duplicates['receipt_id'].astype(str)]

users_receipt_duplicates.head()
```

[103]:
```
   receipt_id duplicates_final  imputed_post_grading_weight  \
0       30689              NaN                       3069.0
1       30687              NaN                       5376.0
2       30685              NaN                          NaN
3       30683              NaN                          NaN
4       30678              NaN                          NaN

   imputed_price_manual  imputed_grade_manual      buyer_receipt        date  \
0                1925.0                   NaN  mill//4//nan//nan  2020-12-01
1                1925.0                   NaN  mill//4//nan//nan  2020-12-01
2                   NaN                   NaN  mill//4//nan//nan  2020-11-30
3                   NaN                   NaN  mill//4//nan//nan  2020-11-28
4                   NaN                   NaN  lr//nan//nan//113  2020-11-27

   Unnamed: 7               price_id
0         NaN  manual_price_id_30689
1         NaN  manual_price_id_30687
2         NaN  manual_price_id_30685
3         NaN  manual_price_id_30683
4         NaN  manual_price_id_30678
```

2.14 Importing buyers table

[104]:
```
buyers = pd.read_excel(f"{notebook_2_output}"+"buyers_complete.xlsx",
                                    #sep=";"
                                   )
```

[105]:
```
buyers.columns
```

[105]:
```
Index(['Unnamed: 0', 'user_id', 'buyer', 'buyer_receipt', 'lng', 'lat',
       'buyer_kecamatan_desa', 'counterfactual_buyer',
       'counterfactual_buyer_kecamatan_desa'],
      dtype='object')
```

[106]:
```
buyers['user_id'] = buyers['user_id'].astype(str).replace('\.0', '', regex=True)
```

3 Creating relevant tables

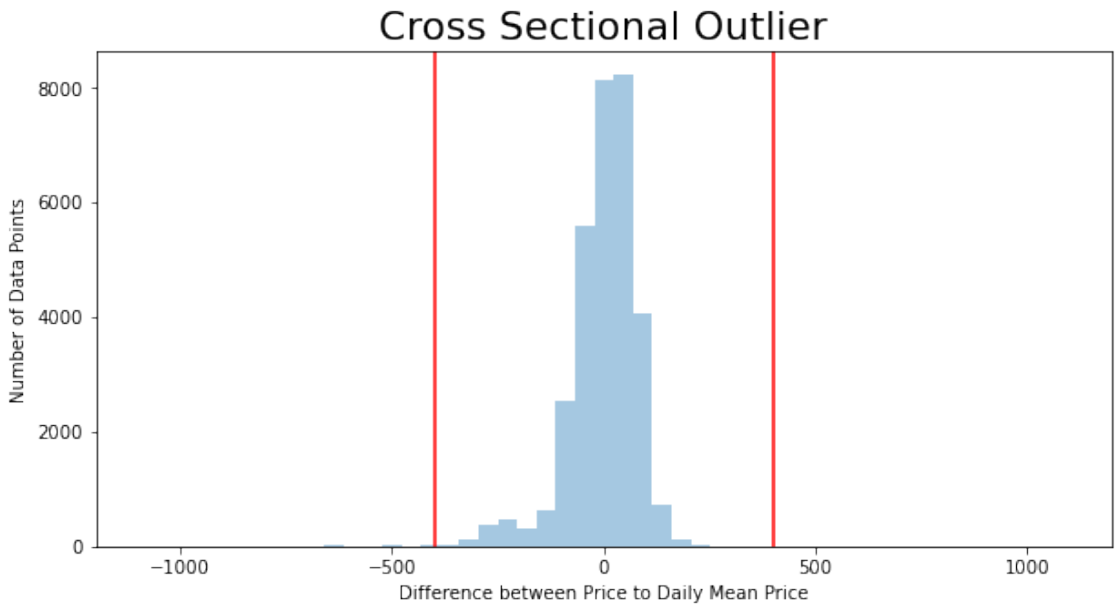
3.1 Defining outlier threshold

[107]:
```
100-(stats.percentileofscore(transactions_sql['price_to_mean_diff'].
↪dropna(),400))
```

[107]:
```
0.09211612985197348
```

[108]:
```
fig, ax = plt.subplots(figsize=(10,5))

plt.rcParams.update({'font.size': 18})
sns.distplot(transactions_sql['price_to_mean_diff'].dropna(), ax = ax,␣
↪kde=False)

plt.axvline(x=-400, color='red')
plt.axvline(x=400, color='red')
plt.xlabel('Difference between Price to Daily Mean Price')
plt.ylabel('Number of Data Points')
plt.xlim(-1200,1200)
plt.title("Cross Sectional Outlier")
plt.savefig(f"{fig_output_path}" +"cross_sectional_outlier.png")
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

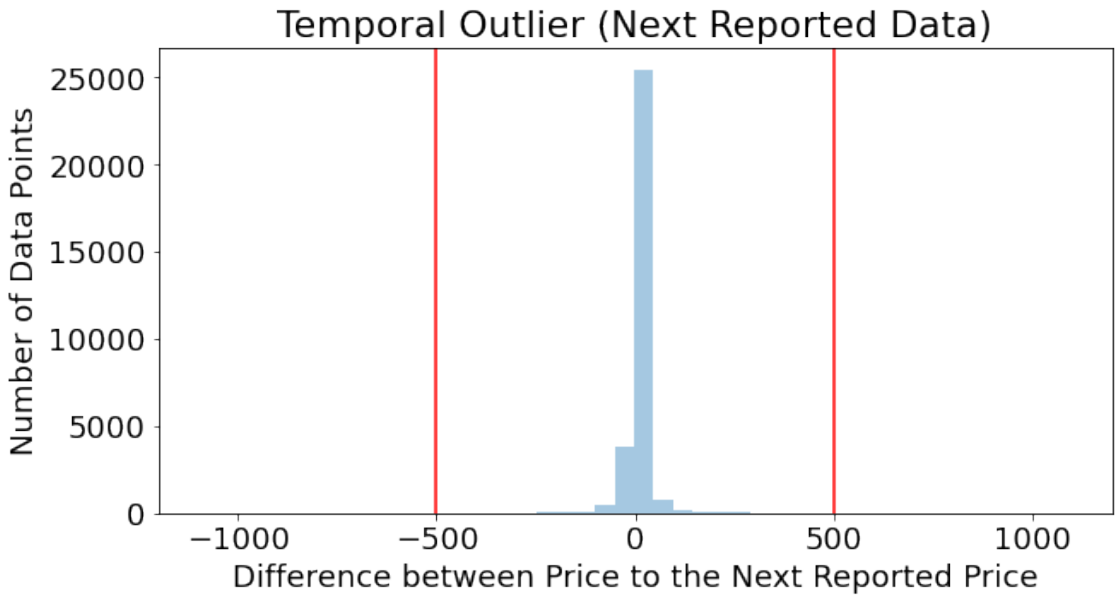


[109]:
```
stats.percentileofscore(transactions_sql['price_to_mean_diff'].dropna(),-400)
```

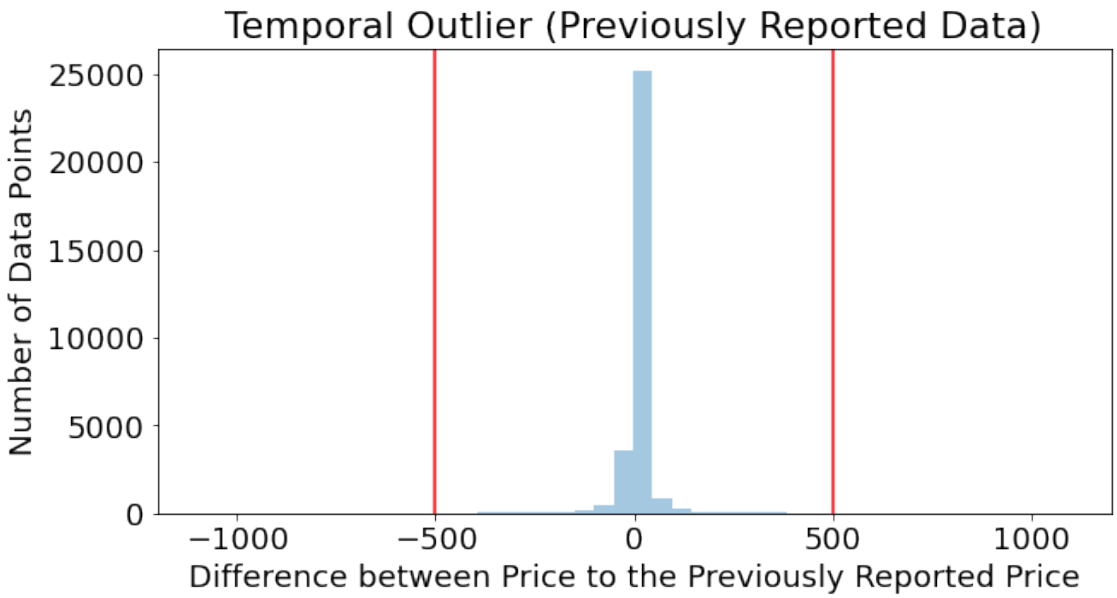
[109]:
```
0.21917286068229463
```

[111]:
```
fig, ax = plt.subplots(figsize=(10,5))
```

```
plt.rcParams.update({'font.size': 18})
sns.distplot(transactions_sql['next_price_diff'].dropna(), ax = ax, kde=False)

plt.axvline(x=-500, color='red')
plt.axvline(x=500, color='red')
plt.xlabel('Difference between Price to the Next Reported Price')
plt.ylabel('Number of Data Points')
plt.xlim(-1200,1200)
plt.title("Temporal Outlier (Next Reported Data)")
plt.savefig(f"{fig_output_path}" +"temporal_outlier_next.png")
```



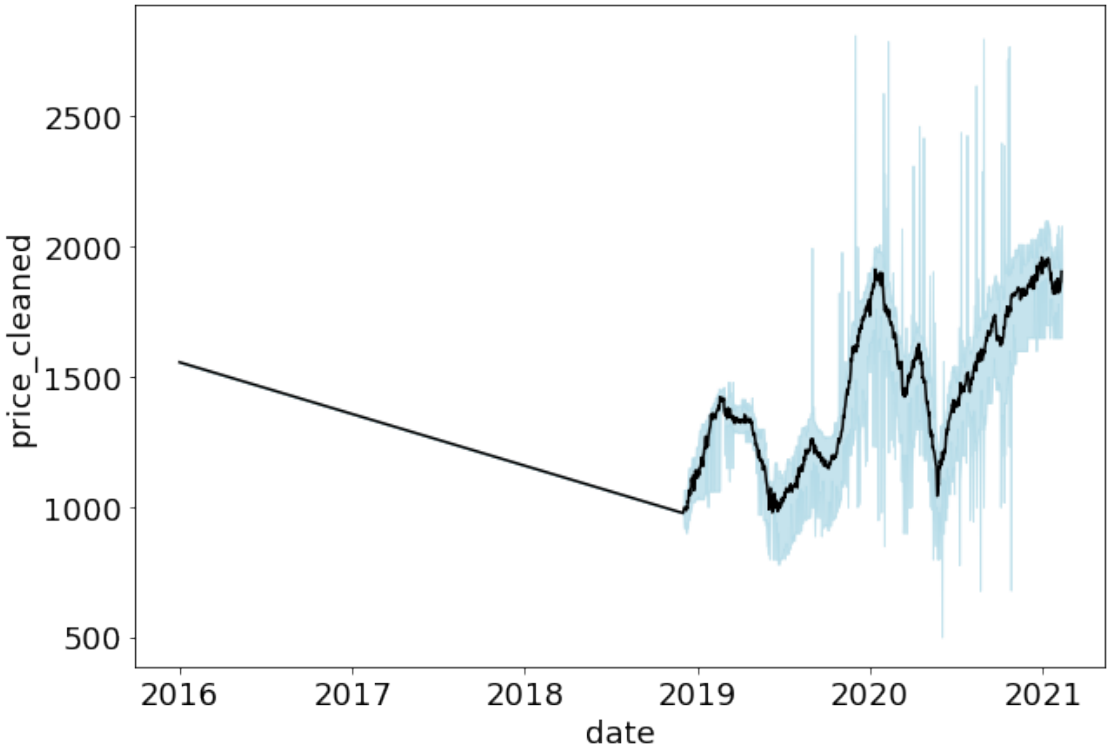
[112]:
```
fig, ax = plt.subplots(figsize=(10,5))

plt.rcParams.update({'font.size': 18})
sns.distplot(transactions_sql['prev_price_diff'].dropna(), ax = ax, kde=False)

plt.axvline(x=-500, color='red')
plt.axvline(x=500, color='red')
plt.xlabel('Difference between Price to the Previously Reported Price')
plt.ylabel('Number of Data Points')
plt.xlim(-1200,1200)
plt.title("Temporal Outlier (Previously Reported Data)")
plt.savefig(f"{fig_output_path}" +"temporal_outlier_prev.png")
```



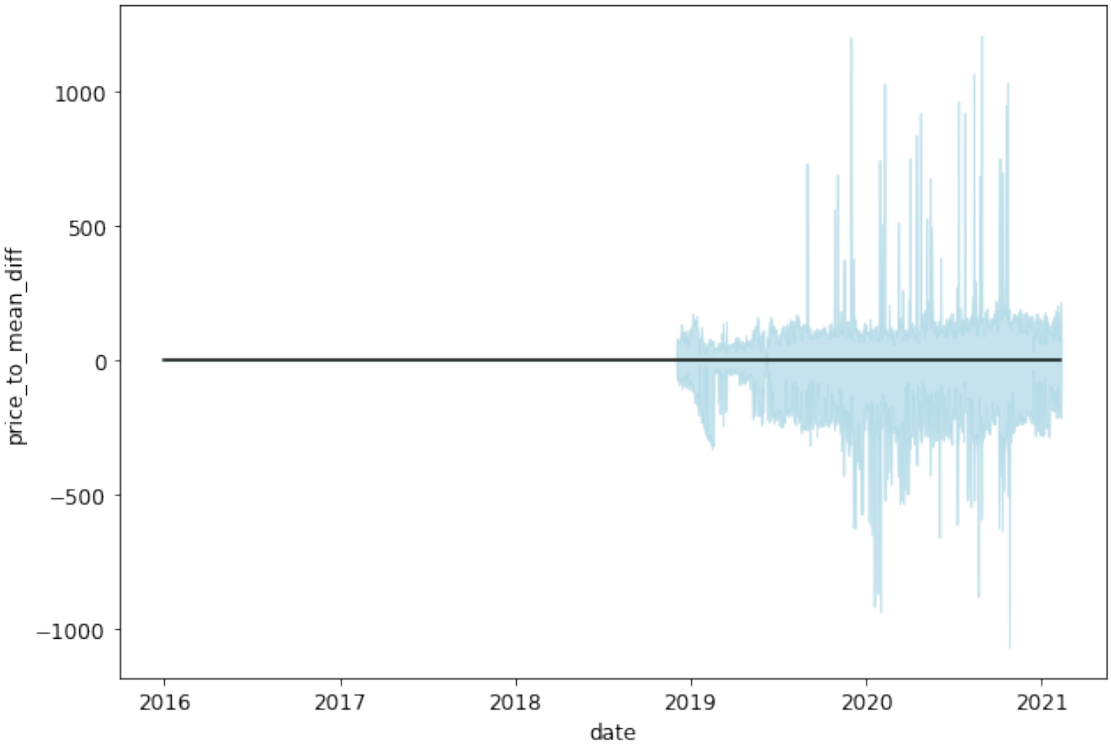
3.2 Remove outlier from transactions_sql

[113]:
```
def plot_min_max_band(data, x, y):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    fig, ax = plt.subplots(figsize=(10,7))

    plt.rcParams.update({'font.size': 12})

    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    plt.fill_between(tmp_df['index'], tmp_df['max'],
                 tmp_df['min'],
                alpha=0.7, color='lightblue')

    plt.xlabel(x)

    plt.ylabel(y)
```

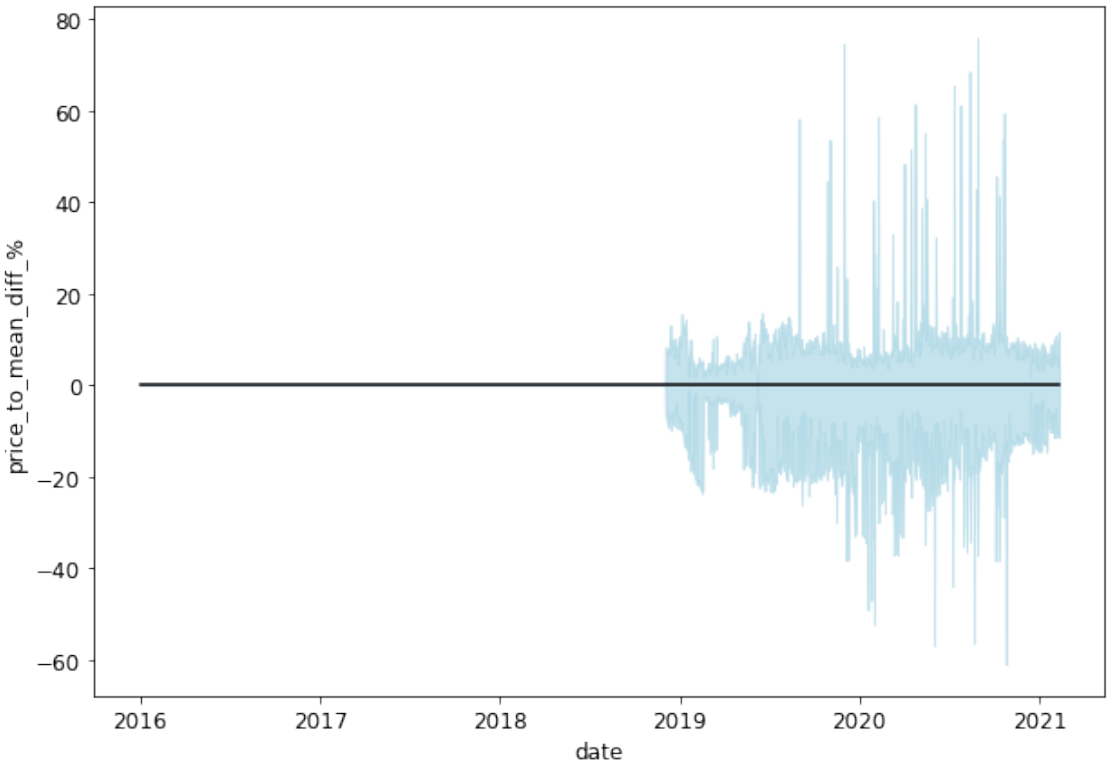
[114]:
```
plot_min_max_band(transactions_sql, 'date', 'price_cleaned')
```



[115]:
```
plot_min_max_band(transactions_sql, 'date', 'price_to_mean_diff')
```



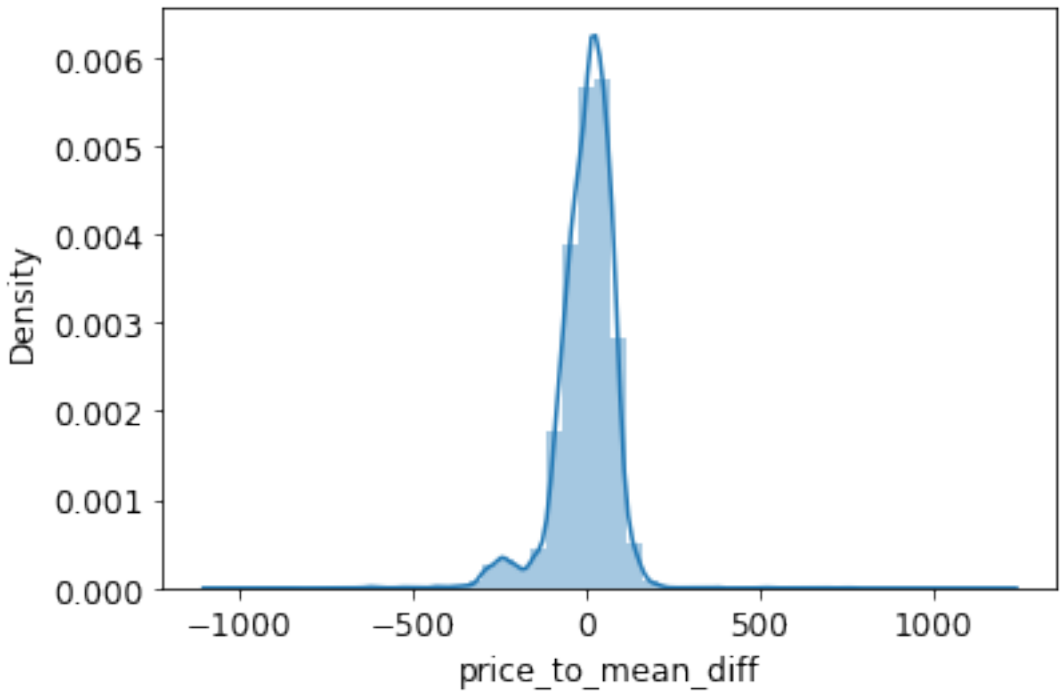
[116]:
```
plot_min_max_band(transactions_sql, 'date', 'price_to_mean_diff_%')
```



[117]:
```
sns.distplot(transactions_sql['price_to_mean_diff'].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

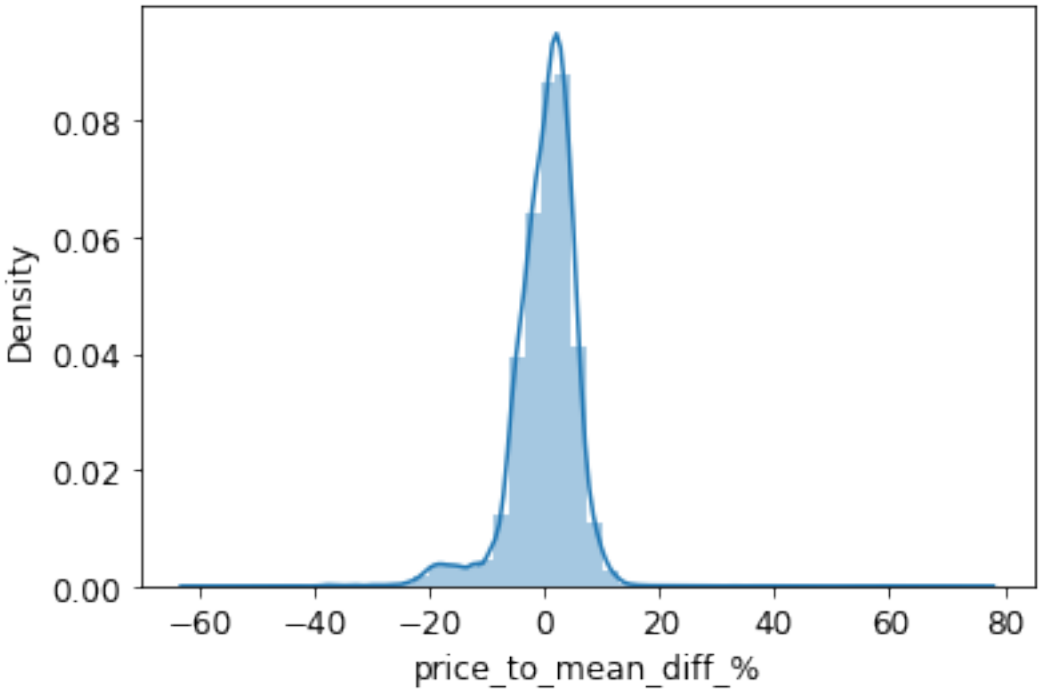
[117]:
```
<AxesSubplot:xlabel='price_to_mean_diff', ylabel='Density'>
```



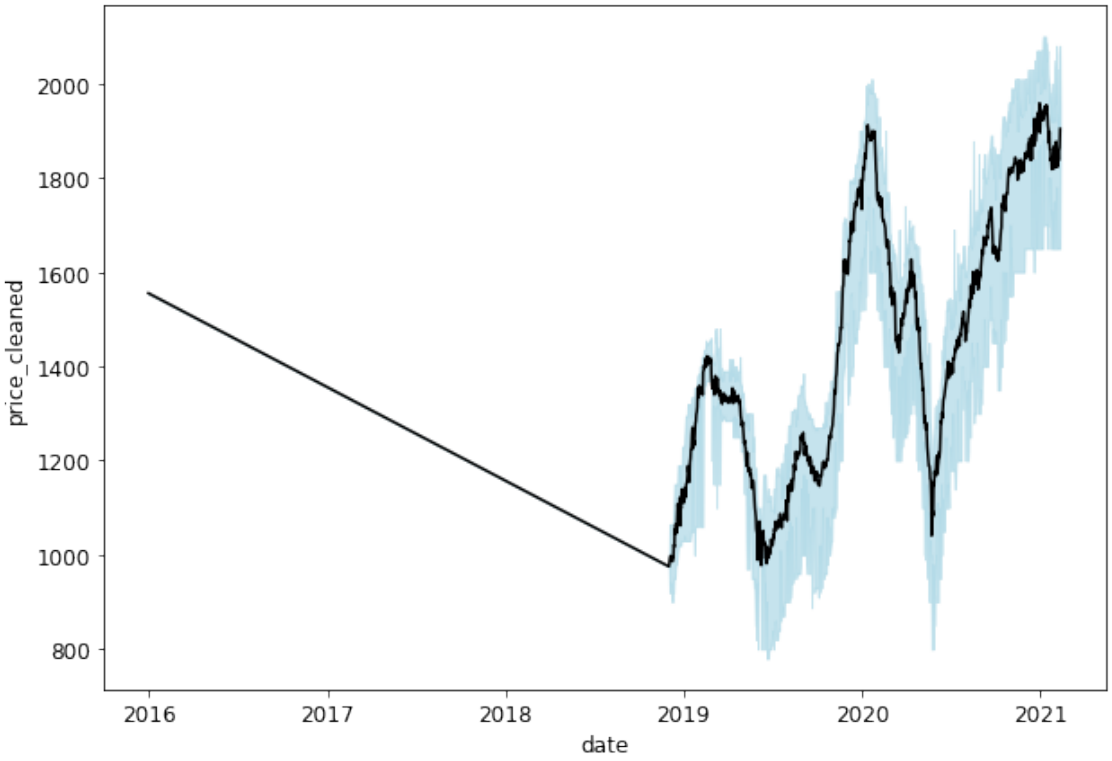
[118]:
```
sns.distplot(transactions_sql['price_to_mean_diff_%'].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[118]:
```
<AxesSubplot:xlabel='price_to_mean_diff_%', ylabel='Density'>
```



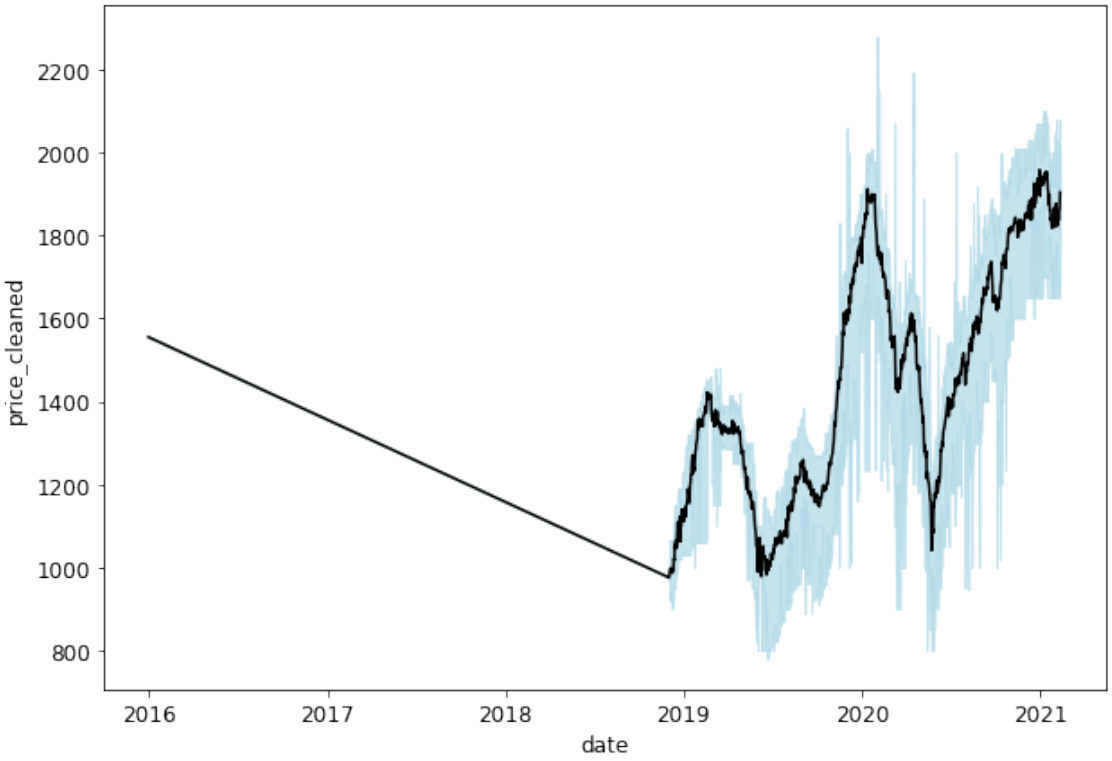
[119]:
```
mask =  (np.absolute(transactions_sql['price_to_mean_diff']) < 300)

plot_min_max_band(transactions_sql[mask]
                  ,
                  'date', 'price_cleaned')
```



[120]:
```
transactions_sql['prev_price_diff_%'] = transactions_sql['prev_price_diff']/
 ↪transactions_sql['price_cleaned']*100
transactions_sql['next_price_diff_%'] = transactions_sql['next_price_diff']/
 ↪transactions_sql['price_cleaned']*100
```

3.2.1 Remove cross sectional outlier

[121]:
```
mask = (#(transactions_sql['price_to_mean_diff_%'] < 40) |
        (np.absolute(transactions_sql['price_to_mean_diff_%']) < 40)
       )

plot_min_max_band(transactions_sql[mask]
                  ,
                  'date', 'price_cleaned')
```

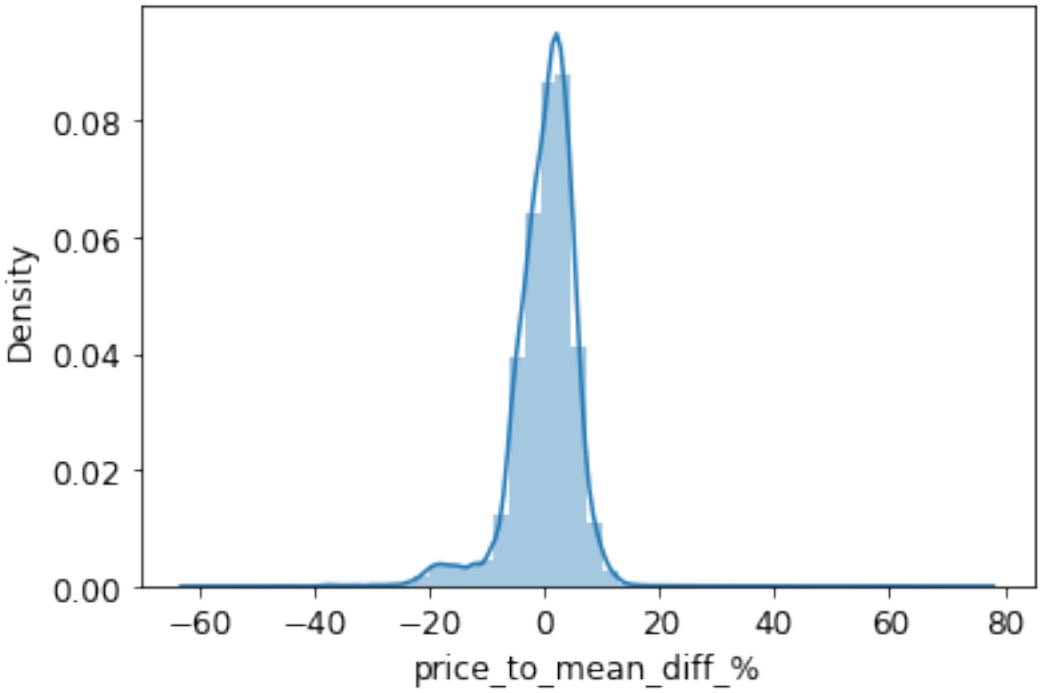


[122]:
```
transcations_sql['price_to_mean_diff_%'] = ((transcations_sql['price_cleaned'] -
                                             ␣
↪transcations_sql['mean_price_daily'])/
                                             ␣
↪transcations_sql['mean_price_daily']
                                             )*100
```

[123]:
```
sns.distplot(transcations_sql['price_to_mean_diff_%'].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[123]:
```
<AxesSubplot:xlabel='price_to_mean_diff_%', ylabel='Density'>
```



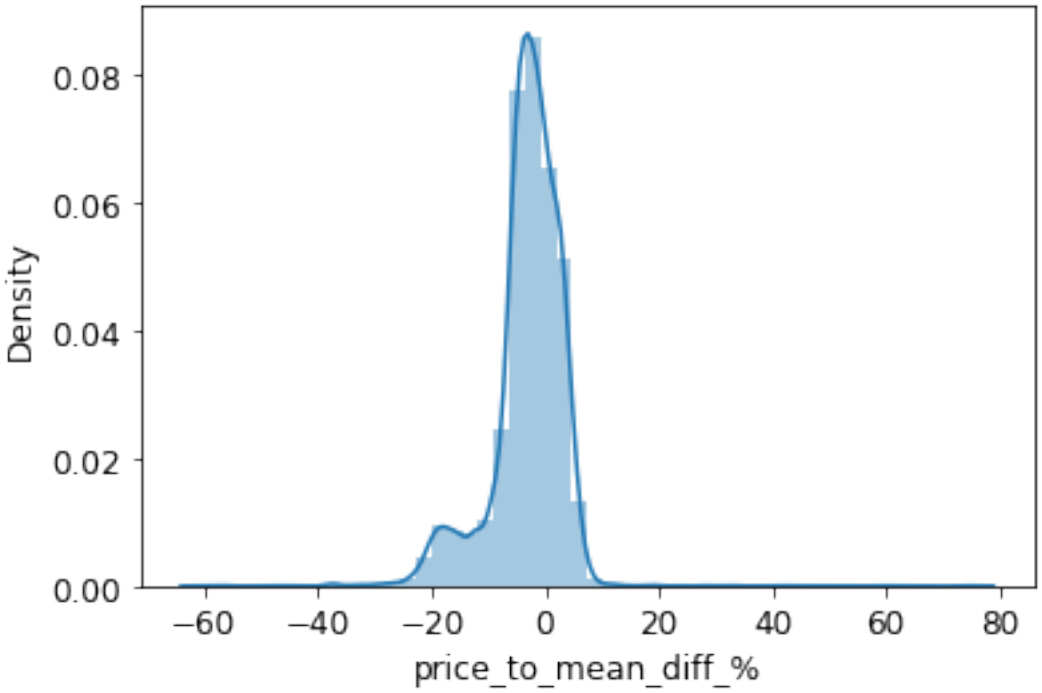
[124]:
```
mask = transactions_sql['price_sharing']

transactions_sql['trans_type'][mask].value_counts()
```

[124]:
```
mill    19401
lr      12275
Name: trans_type, dtype: int64
```

[125]:
```
mask = transactions_sql['trans_type']=='lr'

sns.distplot(transactions_sql['price_to_mean_diff_%'][mask].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

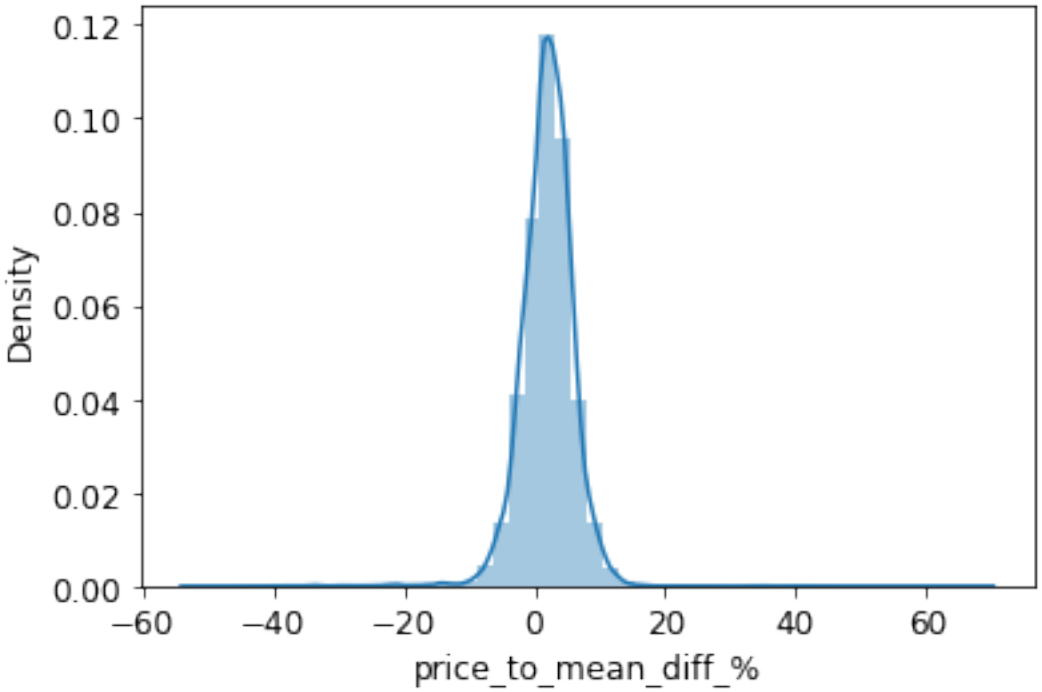
[125]:
```
<AxesSubplot:xlabel='price_to_mean_diff_%', ylabel='Density'>
```



[126]:
```
mask = transactions_sql['trans_type']=='mill'

sns.distplot(transactions_sql['price_to_mean_diff_%'][mask].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[126]:
```
<AxesSubplot:xlabel='price_to_mean_diff_%', ylabel='Density'>
```



[127]:
```
outlier_id = {}
```

[128]:
```
mask = (np.absolute(transactions_sql['price_to_mean_diff_%']) > 40)

for row_id in transactions_sql['id'][mask]:
    outlier_id[row_id] = "Cross Sectional Outlier"
```

3.2.2 Remove 2 sided anomalous neighbors

[129]:
```
transactions_sql.columns
```

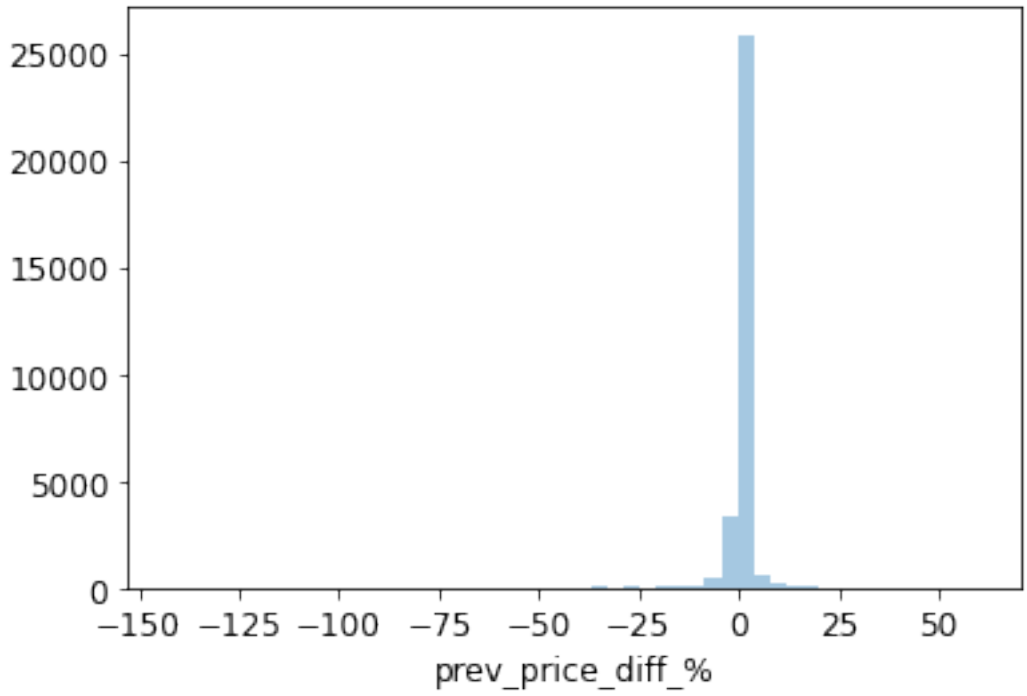
[129]:
```
Index(['id', 'trans_type', 'mill_id', 'lr_id', 'do_id', 'lrm_id', 'user_id',
       'grade', 'bridge', 'day', 'price', 'cash', 'status', 'update_cash',
       'update_price', 'update_grade', 'update_bridge', 'is_do_trans',
       'deleted_at', 'created_at', 'updated_at', 'do_created_at', 'added_by',
       'updated_by', 'net_weight', 'actual_weight', 'client_time_stamp',
       'do_name', 'app_version', 'price_sharing', 'grade_sharing',
       'bridge_sharing', 'cash_sharing', 'row_category_check', 'row_category',
       'date_time', 'date', 'month', 'week', 'biweek', 'triweek', 'quadweek',
       'year', 'year_week', 'year_biweek', 'year_triweek', 'year_quadweek',
       'name', 'phone', 'role', 'buyer', 'route_1', 'route_2', 'route',
       'do_id_1', 'ramp_id_1', 'do/ramp_id', 'buyer_receipt', 'grade_cleaned',
       'price_cleaned', 'average_grade', 'start_window', 'corrected_price',
       'min_price_daily', 'mean_price_daily', 'max_price_daily',
```

```
       'price_to_mean_diff', 'price_to_mean_diff_%', 'prev_price',
       'prev_price_date', 'prev_price_diff', 'next_price', 'next_price_date',
       'next_price_diff', 'price_id', 'prev_price_diff_%',
       'next_price_diff_%'],
      dtype='object')
```

[130]:
```
sns.distplot(transactions_sql['prev_price_diff_%'].dropna(), kde=False)
```

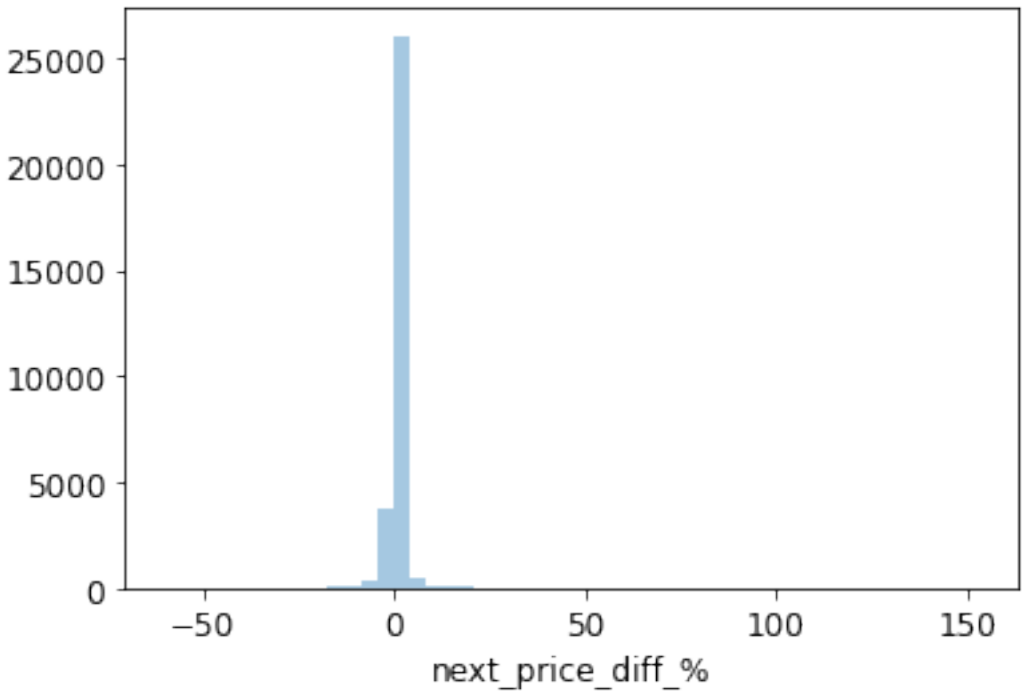
```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[130]:
```
<AxesSubplot:xlabel='prev_price_diff_%'>
```



[131]:
```
sns.distplot(transactions_sql['next_price_diff_%'].dropna(), kde=False)
```

[131]:
```
<AxesSubplot:xlabel='next_price_diff_%'>
```



[132]:
```
mask = ((np.absolute(transactions_sql['prev_price_diff_%']) > 25) &
        (np.absolute(transactions_sql['next_price_diff_%']) > 25) &
        ~(transactions_sql['id'].isin(outlier_id.keys()))
       )

anomalous_neighbor_outlier = transactions_sql[mask]

anomalous_neighbor_outlier['buyer'].value_counts()
```

[132]:
```
lr//nan//nan//54     3
lr//nan//nan//3      3
mill//2//nan//nan    2
mill//17//45//nan    1
mill//1//21//nan     1
mill//6//13//nan     1
mill//4//33//nan     1
mill//3//16//nan     1
lr//nan//nan//51     1
mill//1//nan//nan    1
mill//7//nan//nan    1
mill//2//26//nan     1
lr//nan//nan//43     1
lr//nan//nan//2      1
Name: buyer, dtype: int64
```

[133]:
```
cols = ['id','user_id','buyer','price_to_mean_diff',
        'price_cleaned','prev_price','next_price',
        'date','prev_price_date','next_price_date',
        'prev_price_diff_%','next_price_diff_%',
        'role']

anomalous_neighbor_outlier[cols].sort_values(by='price_cleaned')
```

[133]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	prev_price	next_price	date	prev_price_date	next_price_date
62563	63518	156	mill//6//13//nan	-520.175439	950.0	1570.0	1570.0	2020-07-30	2020-07-30	2020-07-30
64940	65895	124	mill//4//33//nan	-521.388889	1000.0	1570.0	1570.0	2020-08-13	2020-08-13	2020-08-14
67348	68303	97	lr//nan//nan//54	-591.700000	1000.0	1580.0	1580.0	2020-08-28	2020-08-27	2020-08-28
74430	75385	1459	mill//1//21//nan	-624.701493	1000.0	1690.0	1720.0	2020-10-03	2020-10-01	2020-10-03
75988	76943	322	mill//2//nan//nan	-635.610169	1018.0	1725.0	1800.0	2020-10-10	2020-10-10	2020-10-10
44366	45321	610	lr//nan//nan//54	-383.257576	1150.0	1570.0	1590.0	2020-03-30	2020-03-28	2020-03-30
39453	40408	111	lr//nan//nan//51	-463.015152	1185.0	1680.0	1680.0	2020-02-22	2020-02-21	2020-02-22
74649	75604	1460	mill//2//26//nan	-419.035088	1200.0	1655.0	1655.0	2020-10-04	2020-10-03	2020-10-04
76481	77436	294	mill//3//16//nan	-486.000000	1200.0	1775.0	1775.0	2020-10-12	2020-10-10	2020-10-14
44711	45666	675	mill//2//nan//nan	-328.204082	1230.0	1660.0	1640.0	2020-04-01	2020-03-31	2020-04-02
78259	79214	1432	mill//7//nan//nan	-503.603175	1234.0	1780.0	1810.0	2020-10-21	2020-10-21	2020-10-23
39154	40109	321	mill//1//nan//nan	-415.098039	1234.0	1700.0	1770.0	2020-02-20	2020-02-20	2020-02-21
38225	39180	33	lr//nan//nan//3	-439.615385	1260.0	1700.0	1700.0	2020-02-14	2020-02-14	2020-02-14
53536	54491	33	lr//nan//nan//3	380.918367	1560.0	1125.0	1125.0	2020-06-04	2020-06-04	2020-06-04
44176	45131	560	lr//nan//nan//54	71.111111	1570.0	1000.0	1150.0	2020-03-28	2020-03-28	2020-03-30
51265	52220	33	lr//nan//nan//3	375.319149	1580.0	1150.0	1130.0	2020-05-17	2020-05-16	2020-05-18
49583	50538	918	mill//17//45//nan	527.592593	1890.0	1390.0	1390.0	2020-05-06	2020-05-06	2020-05-06
26412	27367	488	lr//nan//nan//43	440.357143	2060.0					
46761	47716	743	lr//nan//nan//2	604.285714	2194.0					

```
26412      1505.0      1505.0  2019-12-01      2019-11-30      2019-12-01
46761      1630.0      1640.0  2020-04-15      2020-04-15      2020-04-15

       prev_price_diff_%  next_price_diff_%          role
62563         -65.263158          65.263158     Middle Man
64940         -57.000000          57.000000     Middle Man
67348         -58.000000          58.000000     Middle Man
74430         -69.000000          72.000000     Middle Man
75988         -69.449902          76.817289     Middle Man
44366         -36.521739          38.260870   Ramp Manager
39453         -41.772152          41.772152  Amprah Farmer
74649         -37.916667          37.916667  Amprah Farmer
76481         -47.916667          47.916667     Middle Man
44711         -34.959350          33.333333     Middle Man
78259         -44.246353          46.677472   Ramp Manager
39154         -37.763371          43.435981     Middle Man
38225         -34.920635          34.920635     Middle Man
53536          27.884615         -27.884615     Middle Man
44176          36.305732         -26.751592  Amprah Farmer
51265          27.215190         -28.481013     Middle Man
49583          26.455026         -26.455026     Middle Man
26412          26.941748         -26.941748     Middle Man
46761          25.706472         -25.250684     Middle Man
```

[134]:
```
for row_id in anomalous_neighbor_outlier['id']:
    outlier_id[row_id] = "Anomalous Neighbor Outlier"
```

[135]:
```
len(outlier_id.keys())
```

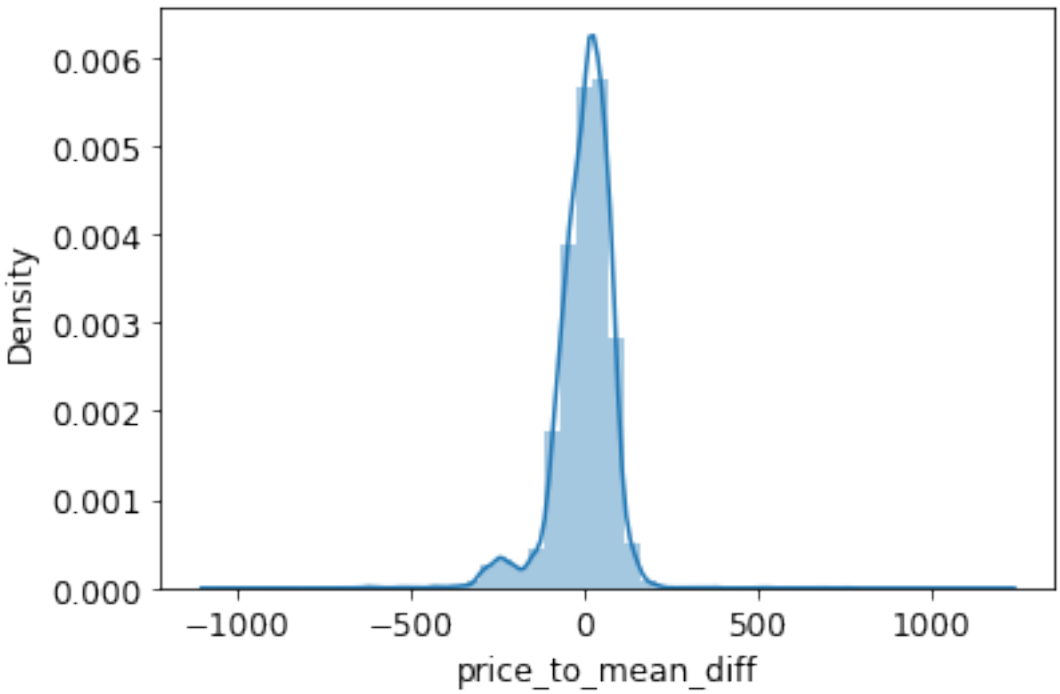
[135]: 49

3.2.3 Remove absolute cross sectional outlier

[136]:
```
sns.distplot(transactions_sql['price_to_mean_diff'].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

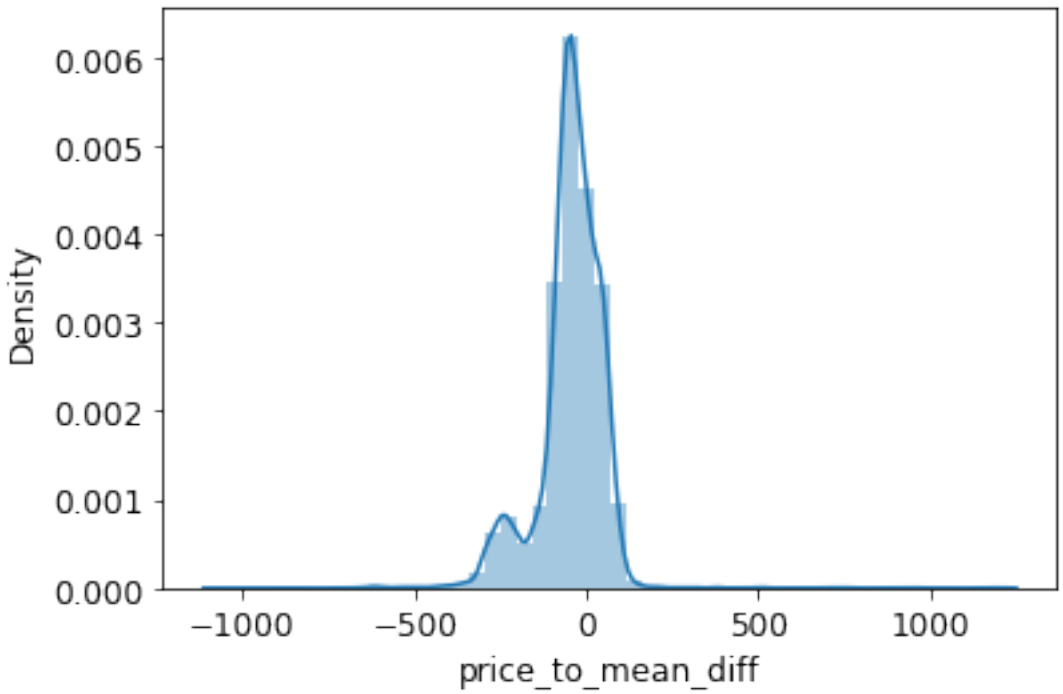
[136]: <AxesSubplot:xlabel='price_to_mean_diff', ylabel='Density'>



[137]:
```
mask = transactions_sql['trans_type']=='lr'

sns.distplot(transactions_sql['price_to_mean_diff'][mask].dropna())
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

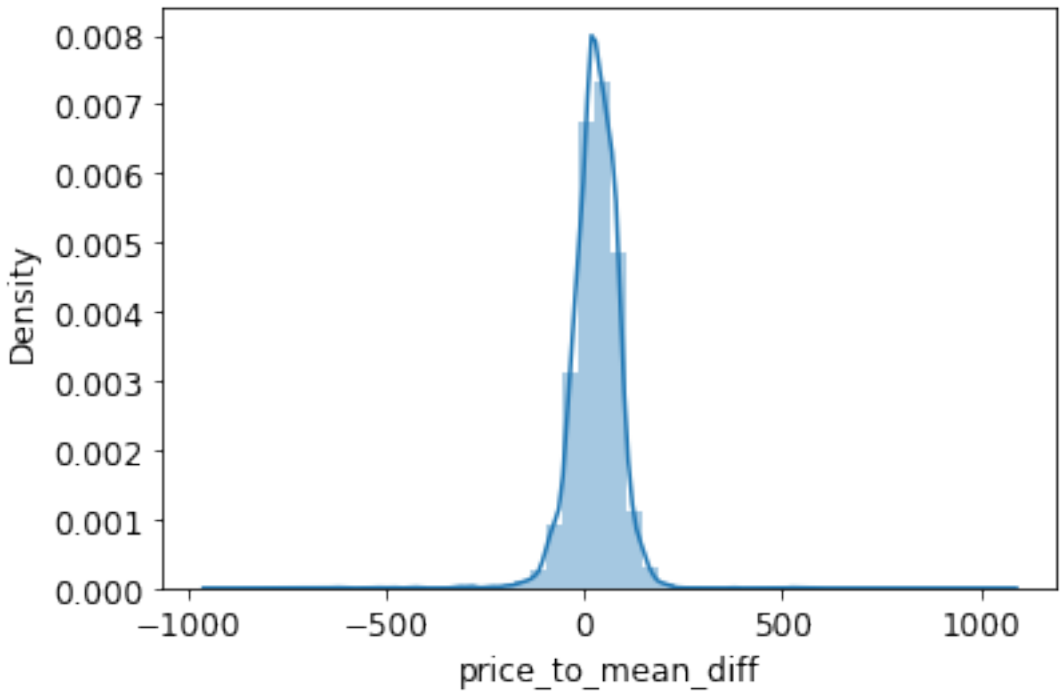
[137]: <AxesSubplot:xlabel='price_to_mean_diff', ylabel='Density'>



[138]:
```
mask = transactions_sql['trans_type']=='mill'

sns.distplot(transactions_sql['price_to_mean_diff'][mask].dropna())
```

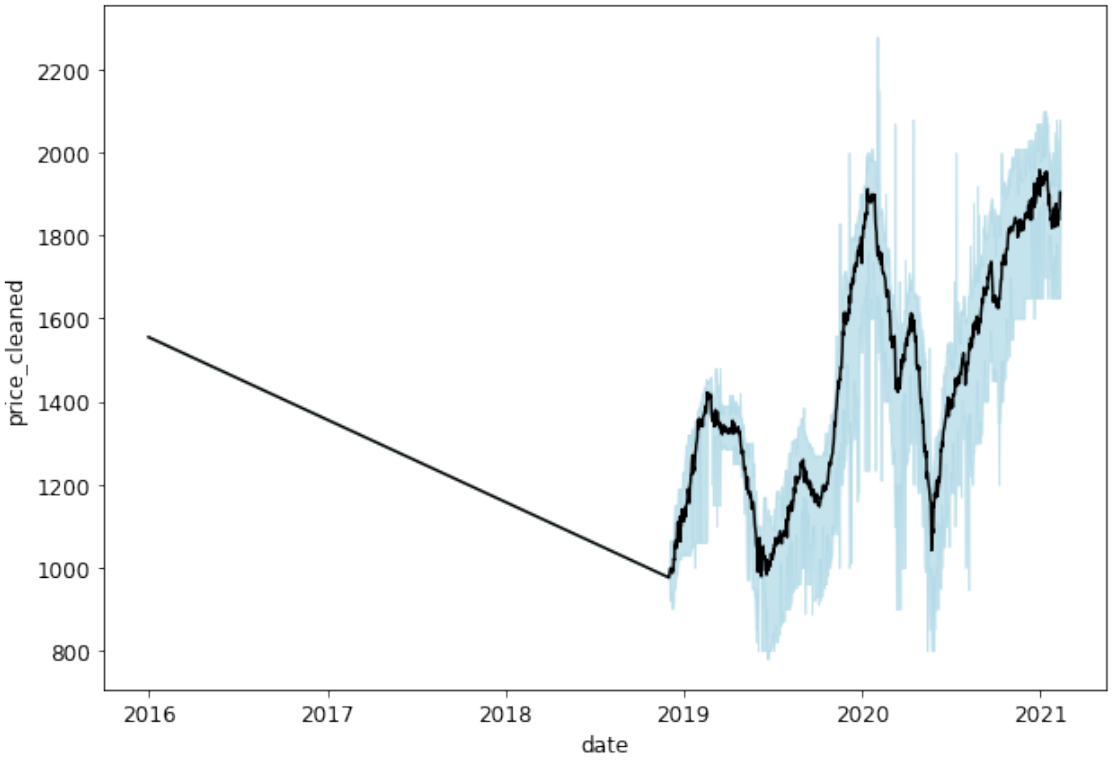
```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[138]:
```
<AxesSubplot:xlabel='price_to_mean_diff', ylabel='Density'>
```



[139]:
```
mask = ~(transactions_sql['id'].isin(outlier_id.keys()))

print(transactions_sql['price_cleaned'][mask].max())

plot_min_max_band(transactions_sql[mask], 'date', 'price_cleaned')
```

```
2280.0
```



[141]:
```
mask = (~(transactions_sql['id'].isin(outlier_id.keys())) &
        ((np.absolute(transactions_sql['price_to_mean_diff']) > 400) |
         (transactions_sql['price_cleaned'] > 2200)
        )
       )

abs_crosssection_outlier = transactions_sql[mask]

abs_crosssection_outlier['buyer'].value_counts()
```

[141]:
```
lr//nan//nan//34      22
lr//nan//nan//54      11
lr//nan//nan//2        4
mill//1//nan//nan      4
mill//47//193//nan     3
mill//31//nan//nan     2
lr//nan//nan//13       2
mill//2//nan//nan      2
lr//nan//nan//20       2
mill//2//15//nan       1
mill//17//45//nan      1
Name: buyer, dtype: int64
```

[142]:
```
abs_crosssection_buyer = list(abs_crosssection_outlier['buyer'].sort_values().
↪unique())
```

[143]:
```
len(abs_crosssection_buyer)
```

[143]: 11

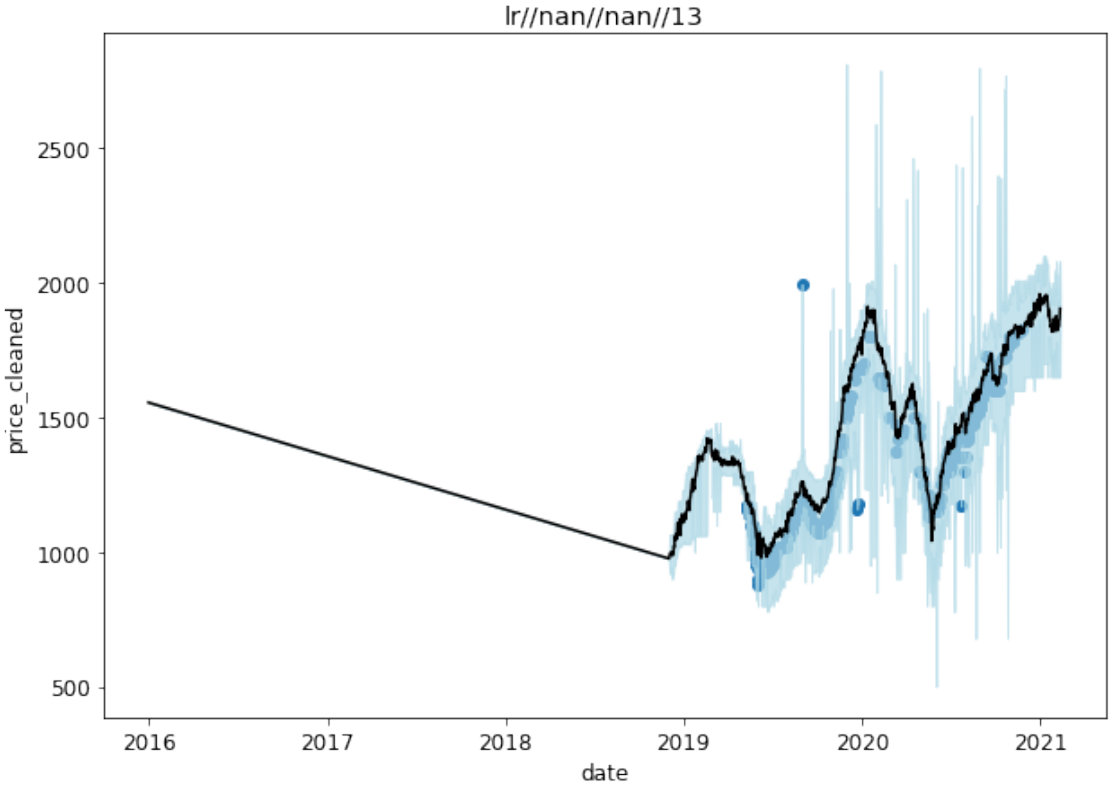
[144]:
```
def get_outlier_consideration_data(data, buyer):
    mask = data['buyer'] == buyer
    cols = ['id','user_id','buyer','price_to_mean_diff',
            'price_cleaned','prev_price','next_price',
            'date','prev_price_date','next_price_date',
            'prev_price_diff_%','next_price_diff_%',
            'role']

    return data[cols][mask].sort_values(by=['price_cleaned'])
```

[145]:
```
def plot_buyer_min_max(data, x, y, buyer):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    print(tmp_df.columns)

    mask_buyer = ((data['buyer'] == buyer) &
                  (data['price_sharing']==True)
                 )

    print(np.sum(mask_buyer))

    buyer_df = data[mask_buyer]

    fig, ax = plt.subplots(figsize=(10,7))

    plt.rcParams.update({'font.size': 12})

    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    plt.scatter(buyer_df[x],buyer_df[y])

    plt.fill_between(tmp_df['index'], tmp_df['max'],
                     tmp_df['min'],
                    alpha=0.7, color='lightblue')
```

```
    plt.title(buyer)

    plt.xlabel(x)

    plt.ylabel(y)
```

[146]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[0])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
371
```



[147]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[0])
```

[147]:

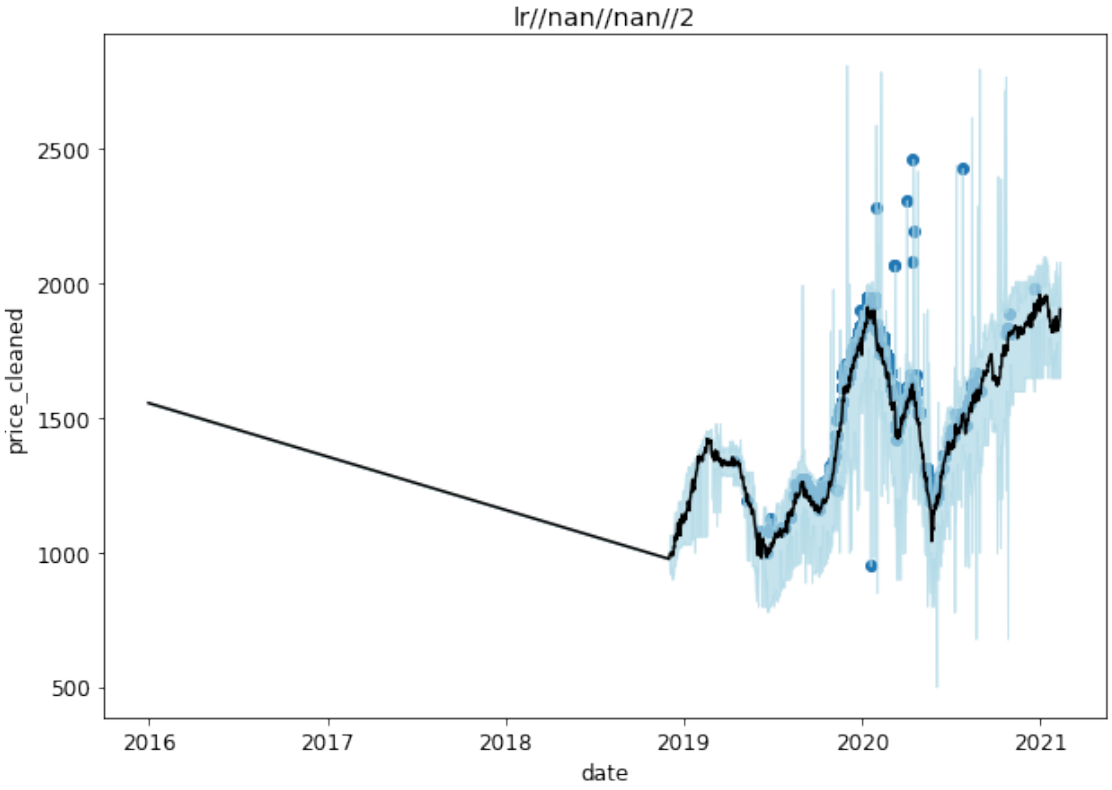
	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
29955	30910	223	lr//nan//nan//13	-572.395833	1160.0	
30338	31293	223	lr//nan//nan//13	-570.900000	1180.0	

	prev_price	next_price	date	prev_price_date	next_price_date	\
29955	1680.0	1180.0	2019-12-22	2019-12-21	2019-12-25	
30338	1160.0	1680.0	2019-12-25	2019-12-22	2019-12-27	

```
       prev_price_diff_%  next_price_diff_%          role
29955         -44.827586           1.724138  Ramp Manager
30338           1.694915          42.372881  Ramp Manager
```

[148]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[1])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
912
```



[149]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[1])
```

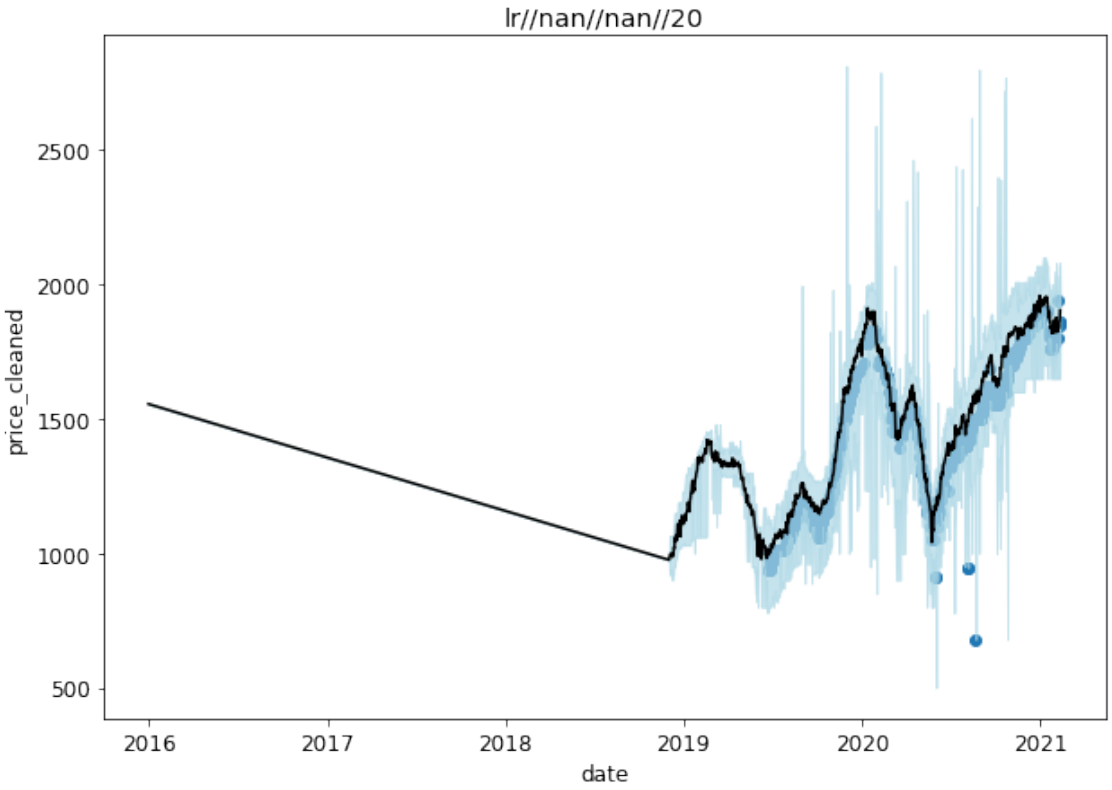
[149]:
```
          id user_id            buyer  price_to_mean_diff  price_cleaned  \
41522  42477     545  lr//nan//nan//2          512.450980         2070.0
41526  42481     545  lr//nan//nan//2          512.450980         2070.0
46555  47510     161  lr//nan//nan//2          454.720000         2080.0
36280  37235     161  lr//nan//nan//2          507.980769         2280.0

       prev_price  next_price        date prev_price_date next_price_date  \
41522      2070.0      2070.0  2020-03-08      2020-03-08      2020-03-08
```

```
41526      2070.0      1615.0  2020-03-08      2020-03-08      2020-03-08
46555      1660.0      1660.0  2020-04-14      2020-04-14      2020-04-14
36280      2280.0      1830.0  2020-02-01      2020-02-01      2020-02-01

       prev_price_diff_%  next_price_diff_%          role
41522           0.000000           0.000000  Amprah Farmer
41526           0.000000         -21.980676  Amprah Farmer
46555          20.192308         -20.192308     Middle Man
36280           0.000000         -19.736842     Middle Man
```

[150]:
```
outlier_id[42477] = "Anomalous Neighbor Outlier"
outlier_id[42481] = "Anomalous Neighbor Outlier"
outlier_id[47510] = "Anomalous Neighbor Outlier"
outlier_id[37235] = "Anomalous Neighbor Outlier"
```

[151]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[2])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
555
```



[152]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[2])
```

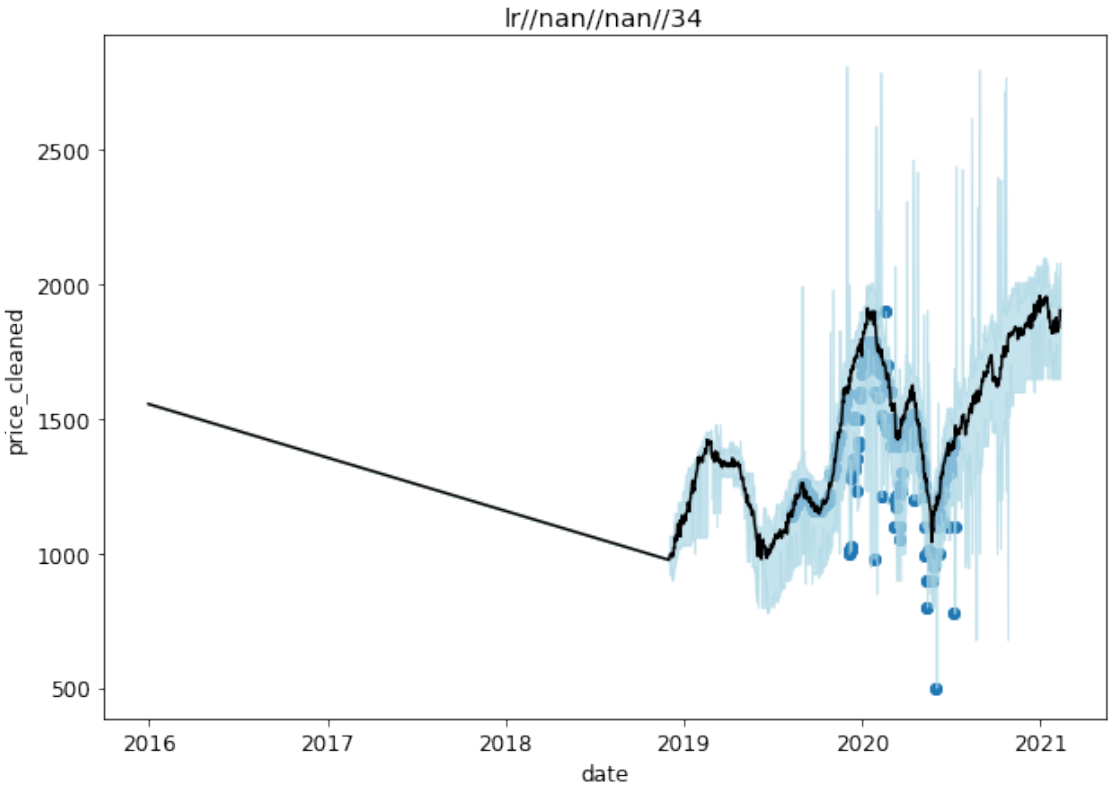
[152]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	prev_price	next_price	date	prev_price_date	next_price_date	prev_price_diff_%	next_price_diff_%	role
63630	64585	986	lr//nan//nan//20	-544.635135	946.0	1420.0	946.0	2020-08-06	2020-08-04	2020-08-06	-50.105708	0.000000	Amprah Farmer
63633	64588	986	lr//nan//nan//20	-544.635135	946.0	946.0	1420.0	2020-08-06	2020-08-06	2020-08-06	0.000000	50.105708	Amprah Farmer

[153]:
```
outlier_id[64585] = "Anomalous Neighbor Outlier"
outlier_id[64588] = "Anomalous Neighbor Outlier"
```

[154]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[3])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
542
```



[155]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[3])
```

[155]:

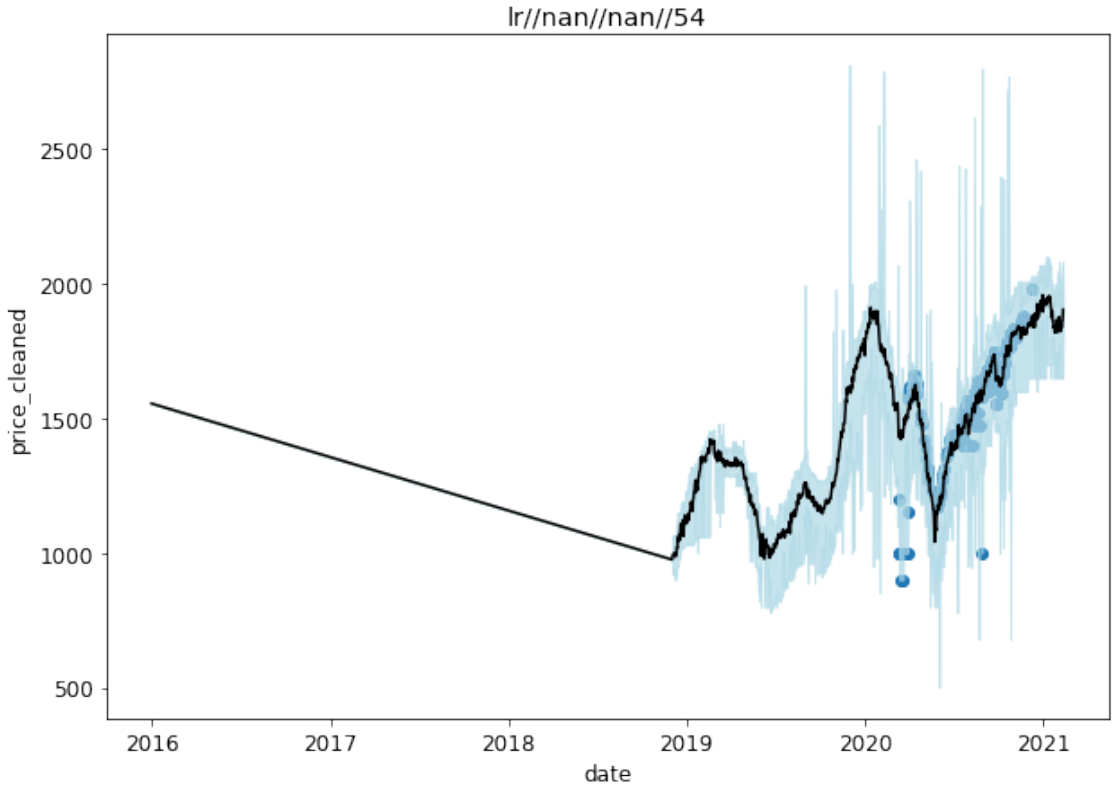
	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
50589	51544	406	lr//nan//nan//34	-428.723404	800.0	
50588	51543	406	lr//nan//nan//34	-428.723404	800.0	
27145	28100	406	lr//nan//nan//34	-621.311475	1000.0	
27146	28101	406	lr//nan//nan//34	-621.311475	1000.0	
27285	28240	406	lr//nan//nan//34	-603.207547	1010.0	
27537	28492	406	lr//nan//nan//34	-626.666667	1010.0	
27538	28493	406	lr//nan//nan//34	-626.666667	1010.0	
27284	28239	406	lr//nan//nan//34	-603.207547	1010.0	
27725	28680	406	lr//nan//nan//34	-625.234375	1025.0	
27726	28681	406	lr//nan//nan//34	-625.234375	1025.0	
41464	42419	406	lr//nan//nan//34	-431.444444	1100.0	
41463	42418	406	lr//nan//nan//34	-431.444444	1100.0	
37571	38526	406	lr//nan//nan//34	-520.600000	1210.0	
37572	38527	406	lr//nan//nan//34	-520.600000	1210.0	
29840	30795	406	lr//nan//nan//34	-496.232143	1234.0	
29841	30796	406	lr//nan//nan//34	-496.232143	1234.0	
27915	28870	406	lr//nan//nan//34	-401.370968	1280.0	
27916	28871	406	lr//nan//nan//34	-401.370968	1280.0	
29225	30180	406	lr//nan//nan//34	-403.135593	1310.0	
29226	30181	406	lr//nan//nan//34	-403.135593	1310.0	
29389	30344	406	lr//nan//nan//34	-403.425926	1320.0	
29388	30343	406	lr//nan//nan//34	-403.425926	1320.0	

	prev_price	next_price	date	prev_price_date	next_price_date	\
50589	800.0	900.0	2020-05-13	2020-05-13	2020-05-13	
50588	1000.0	800.0	2020-05-13	2020-05-11	2020-05-13	
27145	1595.0	1000.0	2019-12-05	2019-12-03	2019-12-05	
27146	1000.0	1010.0	2019-12-05	2019-12-05	2019-12-06	
27285	1010.0	1010.0	2019-12-06	2019-12-06	2019-12-08	
27537	1010.0	1010.0	2019-12-08	2019-12-06	2019-12-08	
27538	1010.0	1025.0	2019-12-08	2019-12-08	2019-12-09	
27284	1000.0	1010.0	2019-12-06	2019-12-05	2019-12-06	
27725	1010.0	1025.0	2019-12-09	2019-12-08	2019-12-09	
27726	1025.0	1280.0	2019-12-09	2019-12-09	2019-12-10	
41464	1100.0	NaN	2020-03-07	2020-03-07	2020-03-08	
41463	1410.0	1100.0	2020-03-07	2020-03-05	2020-03-07	
37571	1700.0	1210.0	2020-02-09	2020-02-08	2020-02-09	
37572	1210.0	1670.0	2020-02-09	2020-02-09	2020-02-10	
29840	1355.0	1234.0	2019-12-21	2019-12-20	2019-12-21	
29841	1234.0	1400.0	2019-12-21	2019-12-21	2019-12-23	
27915	1025.0	1280.0	2019-12-10	2019-12-09	2019-12-10	
27916	1280.0	1300.0	2019-12-10	2019-12-10	2019-12-11	

```
29225      1505.0      1310.0  2019-12-17      2019-12-16      2019-12-17
29226      1310.0      1320.0  2019-12-17      2019-12-17      2019-12-18
29389      1320.0      1350.0  2019-12-18      2019-12-18      2019-12-19
29388      1310.0      1320.0  2019-12-18      2019-12-17      2019-12-18

       prev_price_diff_%  next_price_diff_%         role
50589           0.000000          12.500000  Ramp Manager
50588         -25.000000           0.000000  Ramp Manager
27145         -59.500000           0.000000  Ramp Manager
27146           0.000000           1.000000  Ramp Manager
27285           0.000000           0.000000  Ramp Manager
27537           0.000000           0.000000  Ramp Manager
27538           0.000000           1.485149  Ramp Manager
27284           0.990099           0.000000  Ramp Manager
27725           1.463415           0.000000  Ramp Manager
27726           0.000000          24.878049  Ramp Manager
41464           0.000000                NaN  Ramp Manager
41463         -28.181818           0.000000  Ramp Manager
37571         -40.495868           0.000000  Ramp Manager
37572           0.000000          38.016529  Ramp Manager
29840          -9.805511           0.000000  Ramp Manager
29841           0.000000          13.452188  Ramp Manager
27915          19.921875           0.000000  Ramp Manager
27916           0.000000           1.562500  Ramp Manager
29225         -14.885496           0.000000  Ramp Manager
29226           0.000000           0.763359  Ramp Manager
29389           0.000000           2.272727  Ramp Manager
29388           0.757576           0.000000  Ramp Manager
```

[156]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[4])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
250
```



[157]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[4])
```

[157]:

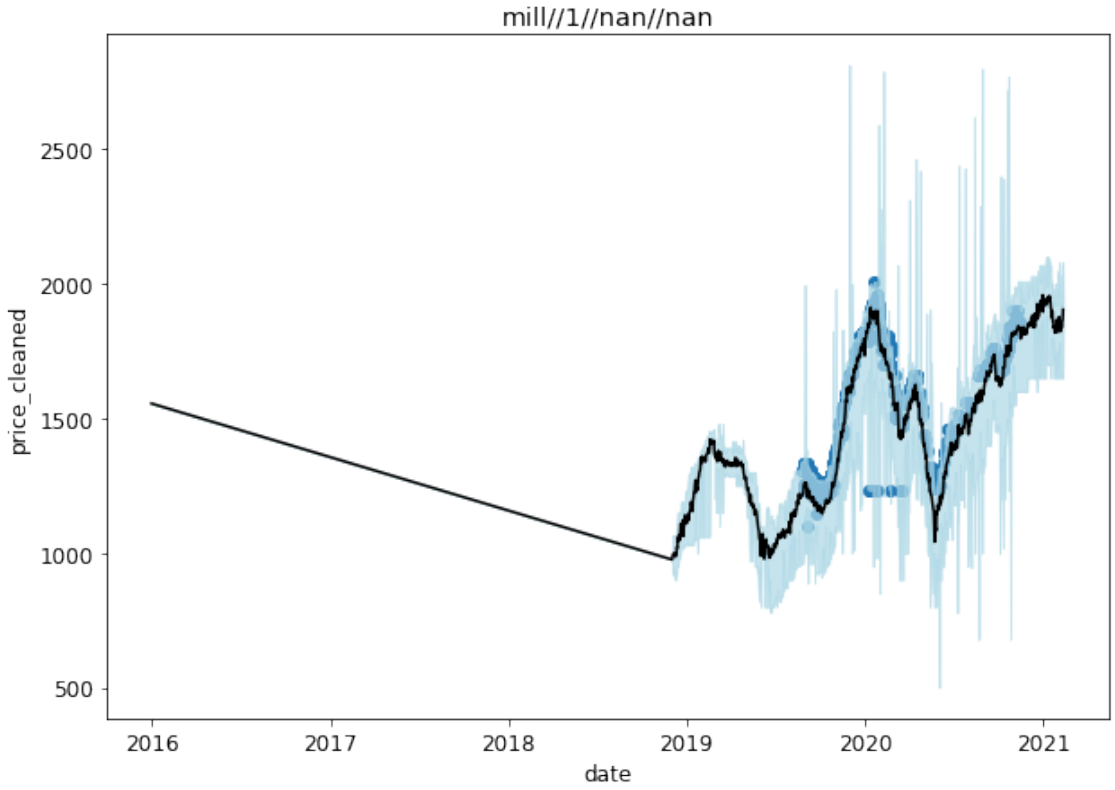
	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
42598	43553	610	lr//nan//nan//54	-522.200000	900.0	
43014	43969	610	lr//nan//nan//54	-532.807018	900.0	
42178	43133	610	lr//nan//nan//54	-428.080000	1000.0	
42179	43134	610	lr//nan//nan//54	-428.080000	1000.0	
42597	43552	610	lr//nan//nan//54	-422.200000	1000.0	
42726	43681	610	lr//nan//nan//54	-455.500000	1000.0	
43163	44118	610	lr//nan//nan//54	-432.807018	1000.0	
43409	44364	610	lr//nan//nan//54	-469.596774	1000.0	
43495	44450	610	lr//nan//nan//54	-480.500000	1000.0	
44007	44962	610	lr//nan//nan//54	-495.175439	1000.0	
44112	45067	610	lr//nan//nan//54	-498.888889	1000.0	

	prev_price	next_price	date	prev_price_date	next_price_date	\
42598	1000.0	1000.0	2020-03-15	2020-03-15	2020-03-16	
43014	1000.0	1000.0	2020-03-19	2020-03-16	2020-03-19	
42178	1200.0	1000.0	2020-03-12	2020-03-11	2020-03-12	
42179	1000.0	1000.0	2020-03-12	2020-03-12	2020-03-15	
42597	1000.0	900.0	2020-03-15	2020-03-12	2020-03-15	

```
42726       900.0       900.0  2020-03-16      2020-03-15      2020-03-19
43163       900.0      1000.0  2020-03-19      2020-03-19      2020-03-21
43409      1000.0      1000.0  2020-03-21      2020-03-19      2020-03-23
43495      1000.0      1000.0  2020-03-23      2020-03-21      2020-03-26
44007      1000.0      1000.0  2020-03-26      2020-03-23      2020-03-28
44112      1000.0      1570.0  2020-03-28      2020-03-26      2020-03-28

       prev_price_diff_%  next_price_diff_%          role
42598         -11.111111          11.111111  Ramp Manager
43014         -11.111111          11.111111  Ramp Manager
42178         -20.000000           0.000000  Ramp Manager
42179           0.000000           0.000000  Ramp Manager
42597           0.000000         -10.000000  Ramp Manager
42726          10.000000         -10.000000  Ramp Manager
43163          10.000000           0.000000  Ramp Manager
43409           0.000000           0.000000  Ramp Manager
43495           0.000000           0.000000  Ramp Manager
44007           0.000000           0.000000  Ramp Manager
44112           0.000000          57.000000  Ramp Manager
```

[158]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[5])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
532
```



[159]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[5])
```

[159]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
32131	33086	321	mill//1//nan//nan	-602.527273	1234.0	
32783	33738	321	mill//1//nan//nan	-616.387755	1234.0	
33263	34218	321	mill//1//nan//nan	-649.634921	1234.0	
35624	36579	321	mill//1//nan//nan	-623.125000	1234.0	

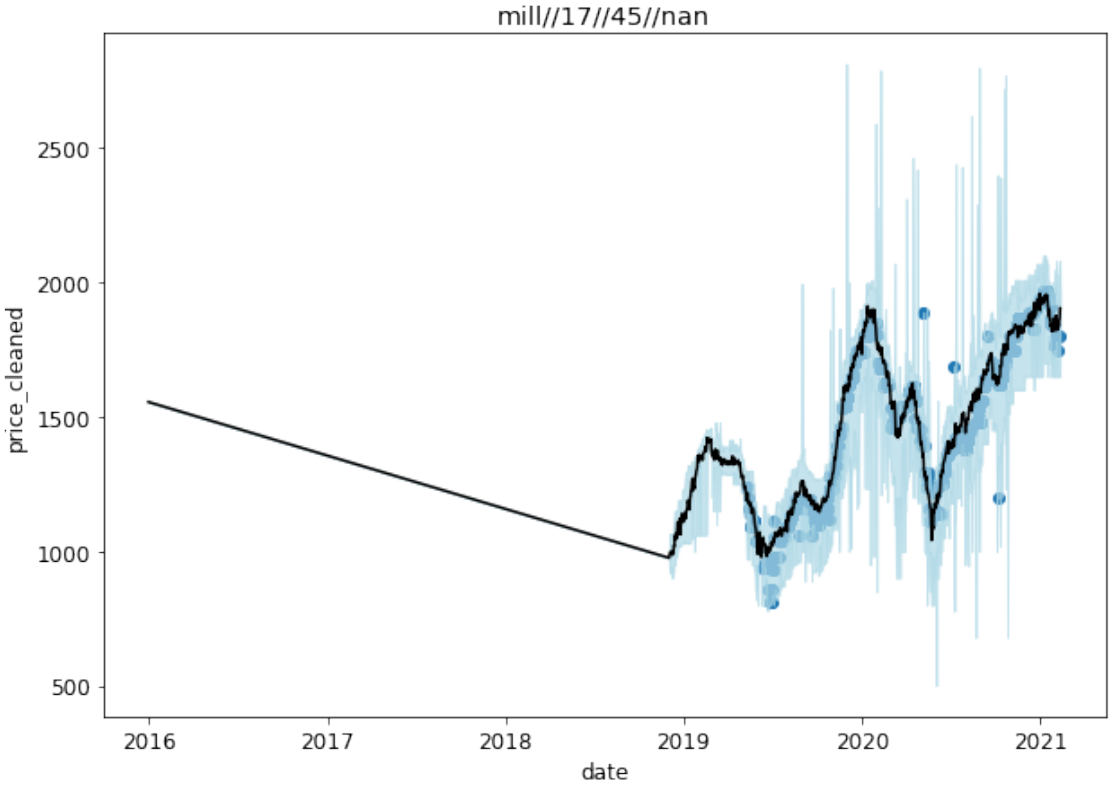
	prev_price	next_price	date	prev_price_date	next_price_date	\
32131	1234.0	1795.0	2020-01-07	2020-01-07	2020-01-07	
32783	1234.0	1910.0	2020-01-11	2020-01-11	2020-01-12	
33263	1234.0	1910.0	2020-01-14	2020-01-14	2020-01-14	
35624	1234.0	1950.0	2020-01-28	2020-01-28	2020-01-28	

	prev_price_diff_%	next_price_diff_%	role
32131	0.0	45.461912	Middle Man
32783	0.0	54.781199	Middle Man
33263	0.0	54.781199	Middle Man
35624	0.0	58.022690	Middle Man

[160]:
```
outlier_id[33086] = "Anomalous Neighbor Outlier"
outlier_id[33738] = "Anomalous Neighbor Outlier"
outlier_id[34218] = "Anomalous Neighbor Outlier"
outlier_id[36579] = "Anomalous Neighbor Outlier"
```

[161]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[6])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
1218
```

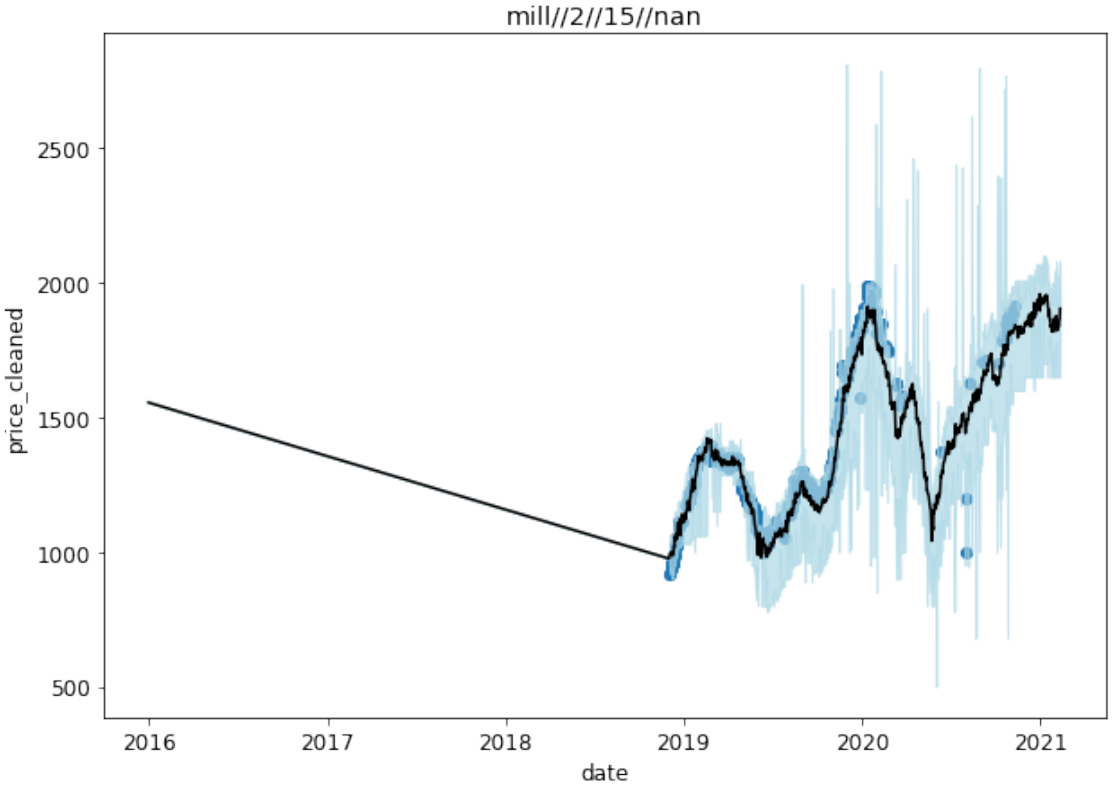


[162]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[6])
```

[162]:
```
          id user_id              buyer  price_to_mean_diff  price_cleaned  \
75347  76302    1456  mill//17//45//nan         -429.516129         1200.0

       prev_price  next_price        date prev_price_date next_price_date  \
75347      1200.0      1620.0  2020-10-07      2020-10-07      2020-10-07

       prev_price_diff_%  next_price_diff_%        role
75347                0.0               35.0  Middle Man
```

[163]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[7])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
396
```



[164]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[7])
```

[164]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
62726	63681	175	mill//2//15//nan	-440.285714	1000.0	

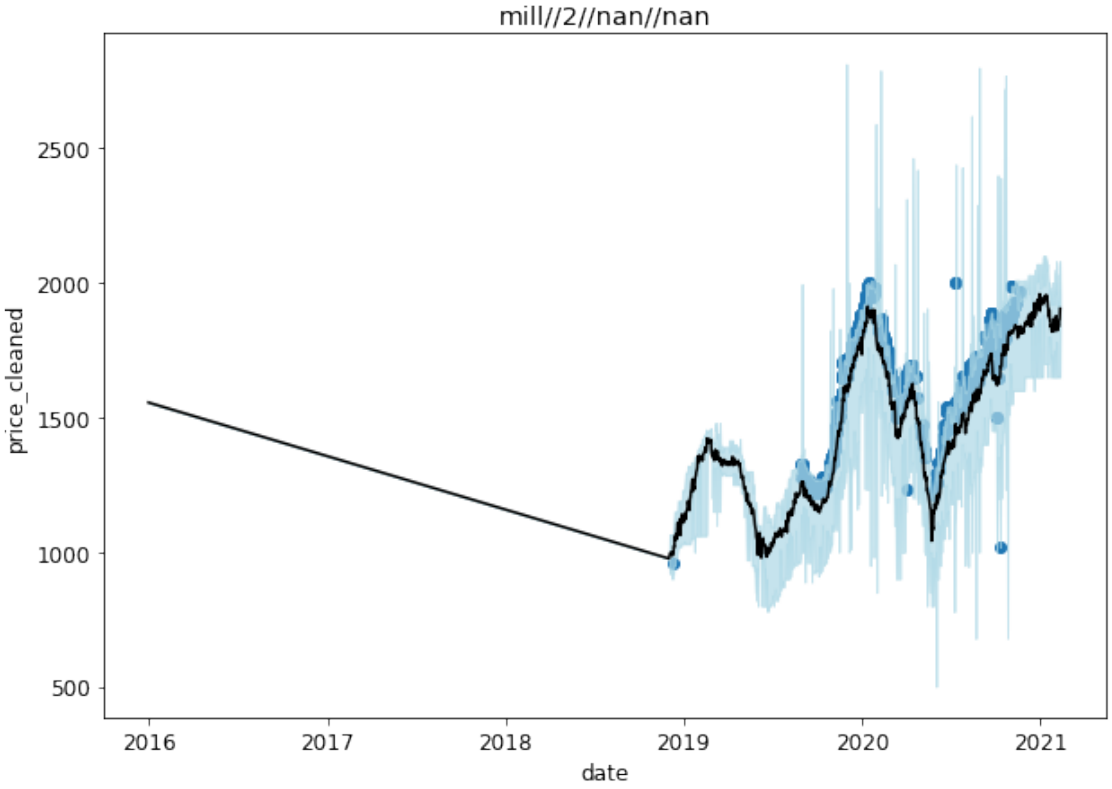
	prev_price	next_price	date	prev_price_date	next_price_date	\
62726	1200.0	1625.0	2020-07-31	2020-07-31	2020-08-09	

	prev_price_diff_%	next_price_diff_%	role
62726	-20.0	62.5	Middle Man

[165]:
```
outlier_id[59957] = "Anomalous Neighbor Outlier"
outlier_id[59958] = "Anomalous Neighbor Outlier"
```

[166]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[8])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
665
```



[167]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[8])
```

[167]:

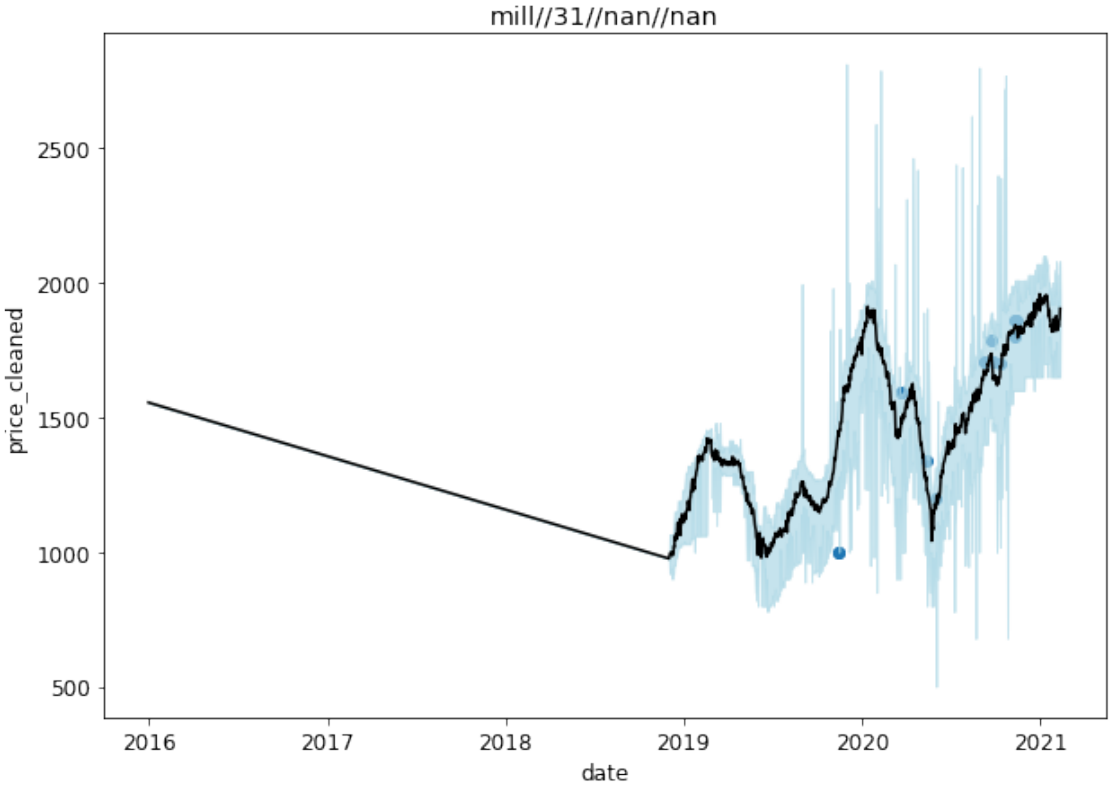
	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
59002	59957	322	mill//2//nan//nan	525.1	2000.0	
59003	59958	322	mill//2//nan//nan	525.1	2000.0	

	prev_price	next_price	date	prev_price_date	next_price_date	\
59002	1490.0	2000.0	2020-07-11	2020-07-06	2020-07-11	
59003	2000.0	1560.0	2020-07-11	2020-07-11	2020-07-11	

	prev_price_diff_%	next_price_diff_%	role
59002	25.5	0.0	Middle Man
59003	0.0	-22.0	Middle Man

[168]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[9])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
13
```



[169]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[9])
```

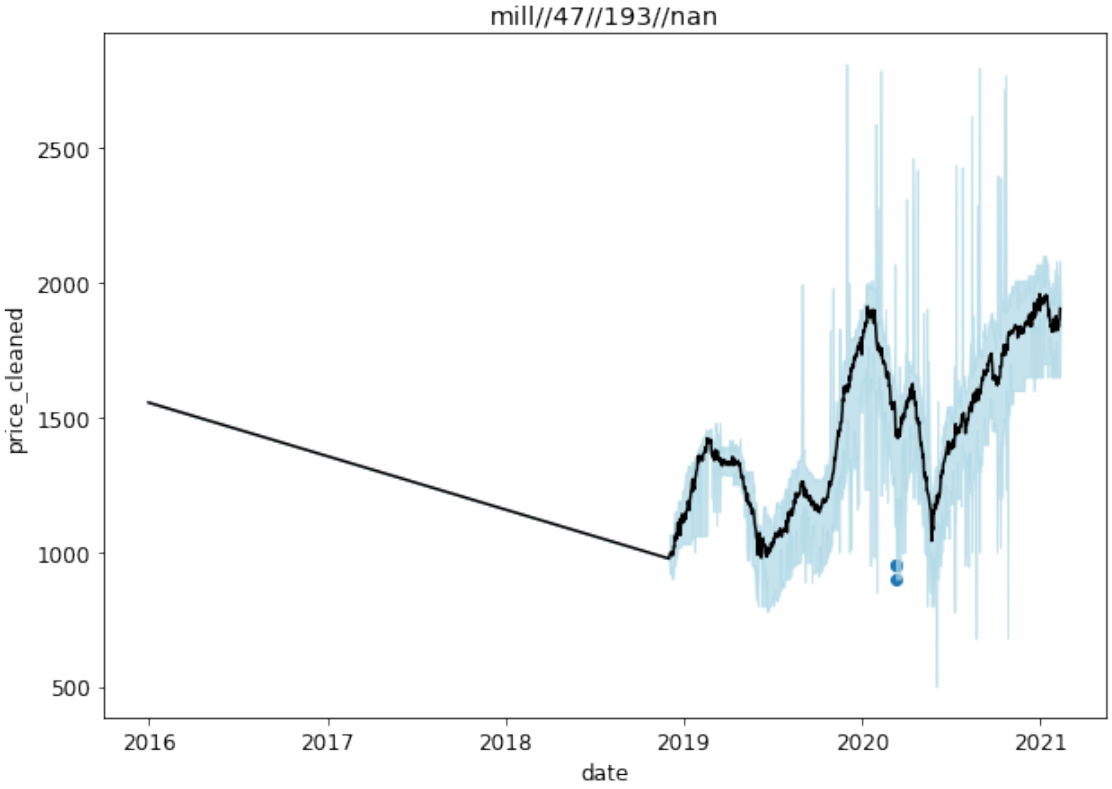
[169]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	prev_price	next_price	date	prev_price_date	next_price_date	prev_price_diff_%	next_price_diff_%	role
23899	24854	539	mill//31//nan//nan	-431.140351	1000.0	NaN	1000.0	2019-11-15	NaN	2019-11-15	NaN	0.0	Amprah Farmer
23900	24855	539	mill//31//nan//nan	-431.140351	1000.0	1000.0	1590.0	2019-11-15	2019-11-15	2020-03-22	0.0	59.0	Amprah Farmer

[170]:
```
outlier_id[37618] = "Anomalous Neighbor Outlier"
```

[171]:
```
plot_buyer_min_max(transactions_sql, 'date', 'price_cleaned',␣
↪abs_crosssection_buyer[10])
```

```
Index(['index', 'min', 'mean', 'max', 'count'], dtype='object')
3
```



[172]:
```
get_outlier_consideration_data(abs_crosssection_outlier,␣
↪abs_crosssection_buyer[10])
```

[172]:

	id	user_id	buyer	price_to_mean_diff	price_cleaned	\
42053	43008	608	mill//47//193//nan	-532.270833	900.0	
42054	43009	608	mill//47//193//nan	-482.270833	950.0	
42177	43132	608	mill//47//193//nan	-478.080000	950.0	

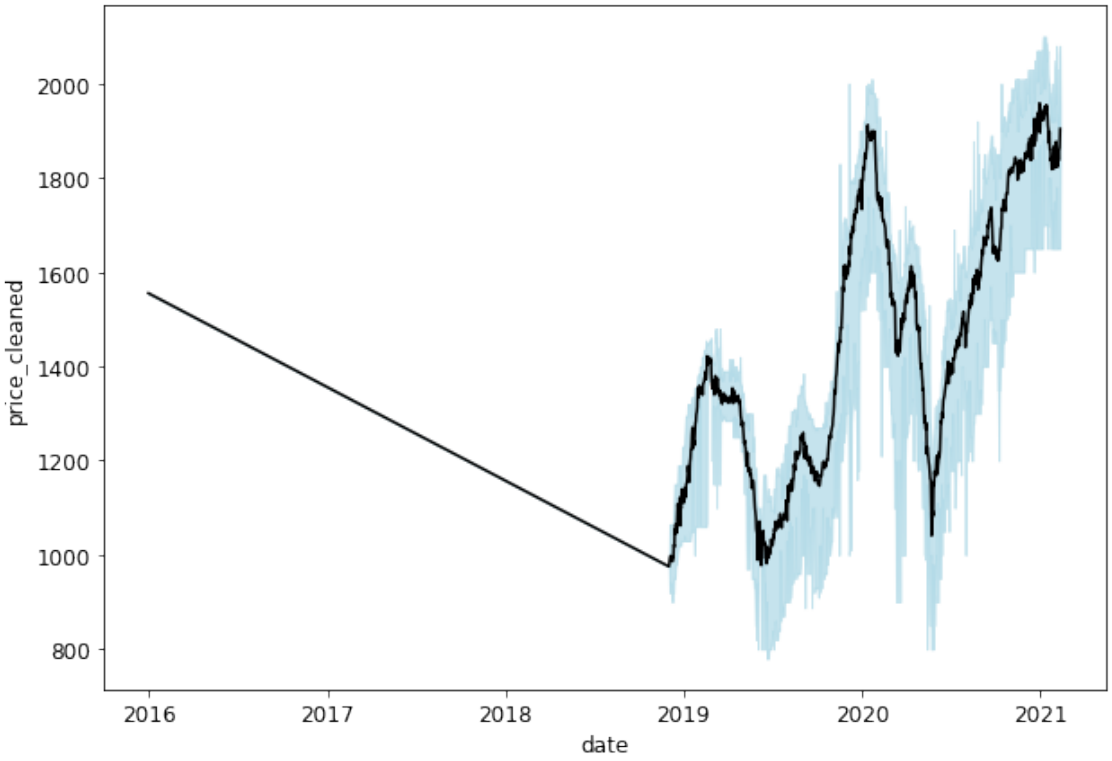
	prev_price	next_price	date	prev_price_date	next_price_date	\
42053	NaN	950.0	2020-03-11	NaN	2020-03-11	
42054	900.0	950.0	2020-03-11	2020-03-11	2020-03-12	
42177	950.0	NaN	2020-03-12	2020-03-11	NaN	

	prev_price_diff_%	next_price_diff_%	role
42053	NaN	5.555556	Farmer
42054	5.263158	0.000000	Farmer

```
42177           0.000000                NaN  Farmer
```

[173]:
```
mask = ~(transactions_sql['id'].isin(outlier_id.keys()))

print(transactions_sql['price_cleaned'][mask].max())

plot_min_max_band(transactions_sql[mask], 'date', 'price_cleaned')
```

```
2100.0
```



[174]:
```
len(outlier_id.keys())
```

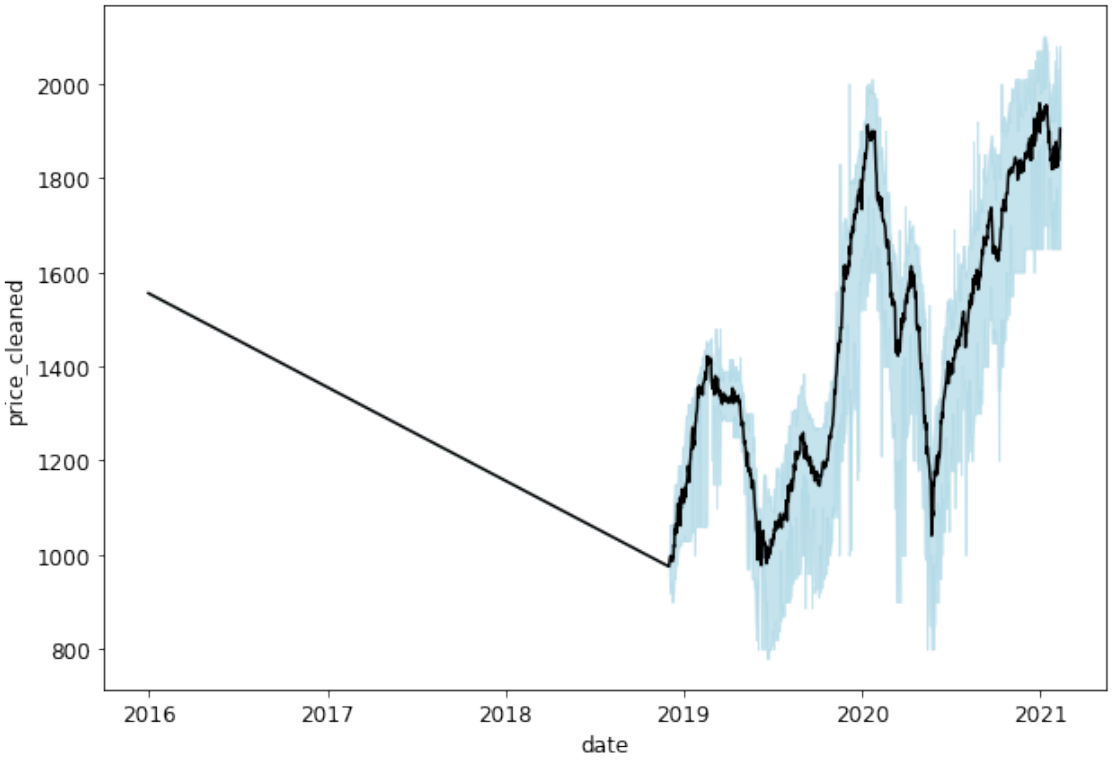
[174]: 62

[175]:
```
sum(~(transactions_sql['price_cleaned'].isna()))
```

[175]: 31482

[176]:
```
mask = ~(transactions_sql['id'].isin(outlier_id.keys()))

print(transactions_sql['price_cleaned'][mask].max())

plot_min_max_band(transactions_sql[mask], 'date', 'price_cleaned')
```

```
2100.0
```

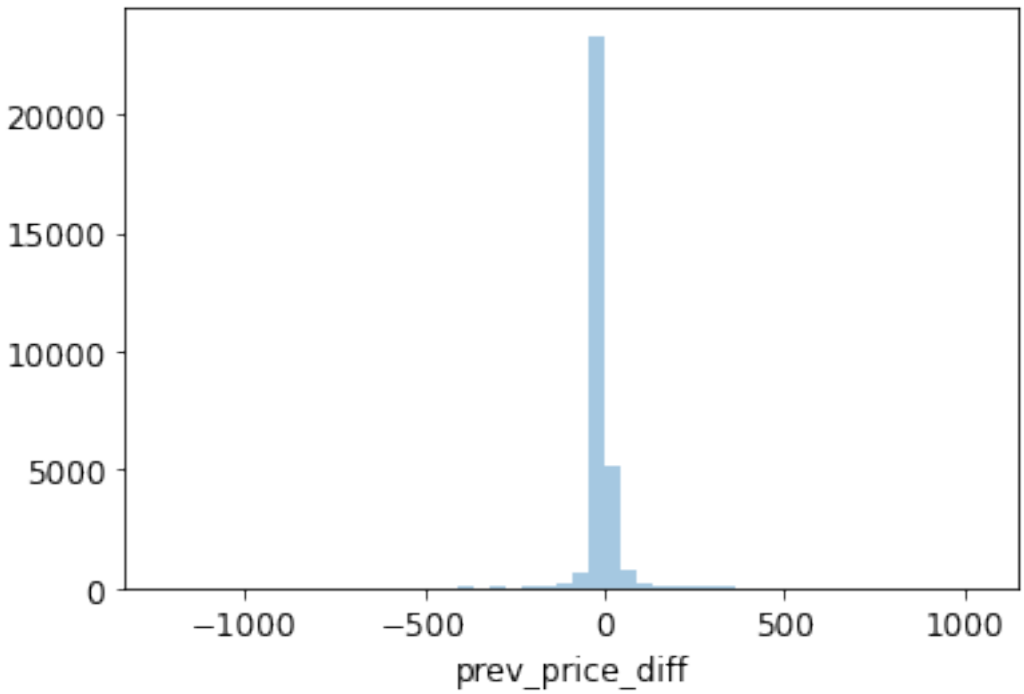


3.2.4 Remove absolute anomalous neighbor

[177]:
```
mask = ~(transactions_sql['id'].isin(outlier_id.keys()))

sns.distplot(transactions_sql['prev_price_diff'][mask].dropna(), kde=False)
```

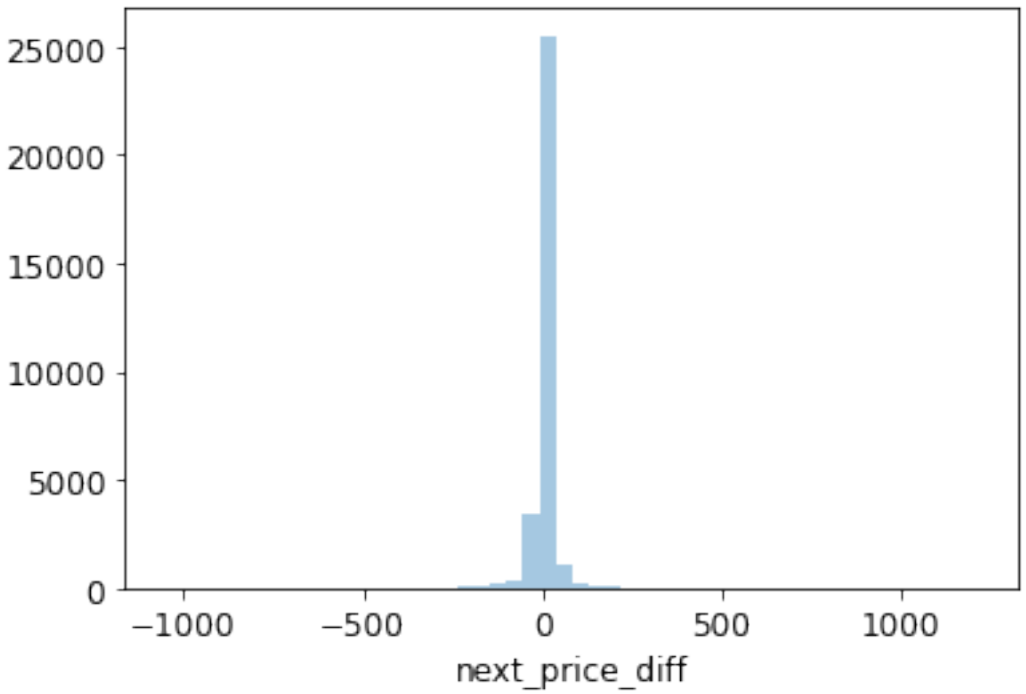
```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[177]:
```
<AxesSubplot:xlabel='prev_price_diff'>
```



[178]:
```
mask = ~(transactions_sql['id'].isin(outlier_id.keys()))

sns.distplot(transactions_sql['next_price_diff'][mask].dropna(), kde=False)
```

[178]:
```
<AxesSubplot:xlabel='next_price_diff'>
```



[179]:
```
mask = (((np.absolute(transactions_sql['prev_price_diff']) > 500) |
         (np.absolute(transactions_sql['next_price_diff']) > 500)
        ) &
        ~(transactions_sql['id'].isin(outlier_id.keys()))
       )

abs_anomalous_neighbor_outlier = transactions_sql[mask]

abs_anomalous_neighbor_outlier['buyer'].value_counts().sum()
```

[179]: 100

[180]:
```
for key in abs_anomalous_neighbor_outlier['id']:
    outlier_id[key] = "Absolute Anomalous Neighbor"
```

[181]:
```
len(outlier_id.keys())
```

[181]: 162

[182]:
```
transactions_sql['outlier'] = transactions_sql['id'].isin(outlier_id.keys())
```

3.3 Create buyers table

[183]:
```
'''
cols = ['user_id','buyer','buyer_receipt','lng', 'lat'
       ]

buyers = pd.concat([lrm[cols].drop_duplicates(),
                    do[cols].drop_duplicates()],
                   ignore_index=True)

buyers = buyers.dropna().reset_index().drop(columns= ['index'])
'''
```

[183]:
```
"\ncols = ['user_id','buyer','buyer_receipt','lng', 'lat'\n       ]\n\nbuyers =
pd.concat([lrm[cols].drop_duplicates(),\n
do[cols].drop_duplicates()],\n                    ignore_index=True)\n\nbuyers =
buyers.dropna().reset_index().drop(columns= ['index'])\n"
```

[184]:
```
'''
def which_polygon(data,row):
    if ((data['lng'].iloc[row]==0) &(data['lat'].iloc[row]==0)):
        output = np.nan
    else :
        point = Point(data['lng'].iloc[row],data['lat'].iloc[row])
        output = []
        for polygon in relevant_village['geometry']:
            output.append(polygon.contains(point))
        chosen_index = np.argmax(output)
        output = relevant_village['kecamatan_desa'].iloc[chosen_index]
    return output
'''
```

[184]:
```
"\ndef which_polygon(data,row):\n    if ((data['lng'].iloc[row]==0)
&(data['lat'].iloc[row]==0)):\n        output = np.nan\n    else :\n
point = Point(data['lng'].iloc[row],data['lat'].iloc[row])\n        output =
[]\n        for polygon in relevant_village['geometry']:\n
output.append(polygon.contains(point))\n        chosen_index =
np.argmax(output)\n        output =
relevant_village['kecamatan_desa'].iloc[chosen_index]\n    return output\n"
```

[185]:
```
#buyers['buyer_kecamatan_desa'] = [which_polygon(buyers,row) for row in buyers.
↪index]
```

[186]:
```
'''
buyers['counterfactual_buyer'] = buyers['buyer']
buyers['counterfactual_buyer_kecamatan_desa'] = buyers['buyer_kecamatan_desa']
'''
```

[186]:
```
"\nbuyers['counterfactual_buyer'] =
buyers['buyer']\nbuyers['counterfactual_buyer_kecamatan_desa'] =
buyers['buyer_kecamatan_desa']\n"
```

[187]:
```
#buyers.head()
```

3.3.1 Merging buyer columns to price_group_user table

[188]:
```
buyers['user_id'].iloc[0]
```

[188]:
```
'202'
```

[189]:
```
print(price_group_user.shape)

cols = ['user_id','buyer','buyer_receipt']

price_group_user = price_group_user.merge(buyers[cols],
                                          left_on='added_by_user_id',
                                          right_on='user_id',
                                          how= 'left'
                                         )

print(price_group_user.shape)
```

```
(763, 11)
(763, 14)
```

[190]:
```
price_group_user.drop(columns='user_id_y', inplace= True)
```

[191]:
```
price_group_user.rename(columns={'user_id_x':'user_id'}, inplace= True)
```

3.4 Creating users_do_ramp table

[192]:
```
last_most_transaction_do_ramp.columns
```

[192]:
```
Index(['id', 'do_name', 'ramp_manager_name', 'mill_name', 'ramp_name',
       'user_id', 'do_id', 'ramp_manager_id', 'mill_id', 'ramp_id',
       'created_at', 'updated_at', 'client_time_stamp', 'deleted_at',
       'trans_type', 'buyer', 'buyer_receipt', 'unknown_do', 'buyer_in_pempem',
       'role', 'source'],
      dtype='object')
```

[193]:
```
cols = ['user_id', 'do_id', 'ramp_manager_id', 'mill_id',␣
↪'ramp_id','buyer','trans_type','source']

users_do_ramp = last_most_transaction_do_ramp[cols].drop_duplicates()
```

```
users_do_ramp.rename(columns= {'ramp_manager_id':'lrm_id','ramp_id':
 ↪'lr_id'},inplace=True)

users_do_ramp.reset_index(inplace=True)

users_do_ramp.drop(['index'], axis = 1, inplace=True)

users_do_ramp.head()
```

[193]:

	user_id	do_id	lrm_id	mill_id	lr_id	buyer	trans_type	source
0	553	21	NaN	1	NaN	mill//1//21//nan	mill	in-app-survey
1	554	45	NaN	17	NaN	mill//17//45//nan	mill	in-app-survey
2	554	185	NaN	33	NaN	mill//33//185//nan	mill	in-app-survey
3	555	21	NaN	1	NaN	mill//1//21//nan	mill	in-app-survey
4	556	NaN	40	NaN	45	lr//nan//nan//45	lr	in-app-survey

[194]:
```
users_do_ramp['source'].value_counts()
```

[194]:
```
in-app-survey    1479
2018 survey        57
Name: source, dtype: int64
```

[195]:
```
users_do_ramp['buyer_receipt'] = (users_do_ramp['trans_type'] + '//' +
                                  users_do_ramp['mill_id'].astype(str) + '//' +
                                  'nan' + '//' +
                                  users_do_ramp['lr_id'].astype(str)
                                 )
```

[196]:
```
users_do_ramp[['lrm_id','lr_id']].drop_duplicates()
```

[196]:

	lrm_id	lr_id
0	NaN	NaN
4	40	45
27	2	2
40	10	13
41	4	4
…	…	…
1399	102	125
1456	99	121

```
1460    107   137
1466     34    39
1467     72    81

[61 rows x 2 columns]
```

[197]:
```
transactions_sql.columns
```

[197]:
```
Index(['id', 'trans_type', 'mill_id', 'lr_id', 'do_id', 'lrm_id', 'user_id',
       'grade', 'bridge', 'day', 'price', 'cash', 'status', 'update_cash',
       'update_price', 'update_grade', 'update_bridge', 'is_do_trans',
       'deleted_at', 'created_at', 'updated_at', 'do_created_at', 'added_by',
       'updated_by', 'net_weight', 'actual_weight', 'client_time_stamp',
       'do_name', 'app_version', 'price_sharing', 'grade_sharing',
       'bridge_sharing', 'cash_sharing', 'row_category_check', 'row_category',
       'date_time', 'date', 'month', 'week', 'biweek', 'triweek', 'quadweek',
       'year', 'year_week', 'year_biweek', 'year_triweek', 'year_quadweek',
       'name', 'phone', 'role', 'buyer', 'route_1', 'route_2', 'route',
       'do_id_1', 'ramp_id_1', 'do/ramp_id', 'buyer_receipt', 'grade_cleaned',
       'price_cleaned', 'average_grade', 'start_window', 'corrected_price',
       'min_price_daily', 'mean_price_daily', 'max_price_daily',
       'price_to_mean_diff', 'price_to_mean_diff_%', 'prev_price',
       'prev_price_date', 'prev_price_diff', 'next_price', 'next_price_date',
       'next_price_diff', 'price_id', 'prev_price_diff_%', 'next_price_diff_%',
       'outlier'],
      dtype='object')
```

3.5 Create special_price_daily table

[198]:
```
buyers.head()
```

[198]:
```
   Unnamed: 0 user_id            buyer    buyer_receipt         lng       lat  \
0           0     202  lr//nan//nan//1  lr//nan//nan//1  102.668298 -0.889027
1           1     203  lr//nan//nan//2  lr//nan//nan//2  102.514567 -0.734371
2           2     204  lr//nan//nan//3  lr//nan//nan//3  102.423992 -0.628439
3           3     205  lr//nan//nan//4  lr//nan//nan//4  102.412089 -0.551158
4           4     218  lr//nan//nan//5  lr//nan//nan//5  102.257117 -0.658824

         buyer_kecamatan_desa counterfactual_buyer  \
0          KEMUNING__KERITANG      lr//nan//nan//1
1     BATANG GANSAL__SEBERIDA      lr//nan//nan//2
2   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//3
3   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//4
4  BATANG CENAKU__BUKIT LIPAI      lr//nan//nan//5

  counterfactual_buyer_kecamatan_desa
0                  KEMUNING__KERITANG
```

```
1             BATANG GANSAL__SEBERIDA
2           SEBERIDA__PANGKALAN KASAI
3           SEBERIDA__PANGKALAN KASAI
4          BATANG CENAKU__BUKIT LIPAI
```

[199]:
```
price_group_price.head()
```

[199]:
```
  price_group_price_id price_group_id added_by_user_id  \
0                    1              1              608
1                    2              2              608
2                    3              1              608
3                    4              3              610
4                    5              4              610

   added_by_backend_user_id       day  price  published  \
0                       NaN  20200312   1100          1
1                       NaN  20200312   1000          1
2                       NaN  20200312   1150          1
3                       NaN  20200312   1400          1
4                       NaN  20200312   1300          1

       client_time_stamp             created_at             updated_at  \
0  2020-03-11 19:08:39.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0
1  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0
2  2020-03-11 19:10:02.0  2020-03-11 19:10:05.0  2020-03-13 07:09:51.0
3  2020-03-11 20:09:17.0  2020-03-11 20:09:27.0  2020-03-15 20:41:01.0
4  2020-03-11 20:11:05.0  2020-03-11 20:11:11.0  2020-03-15 20:41:01.0

              deleted_at        date
0  2020-03-13 07:09:51.0  2020-03-12
1  2020-03-13 07:09:51.0  2020-03-12
2  2020-03-13 07:09:51.0  2020-03-12
3  2020-03-15 20:41:01.0  2020-03-12
4  2020-03-15 20:41:01.0  2020-03-12
```

[200]:
```
print(price_group_price.shape)

price_group_price = price_group_price.merge(buyers,
                                            left_on= 'added_by_user_id',
                                            right_on='user_id',
                                            how='left'
                                           )

print(price_group_price.shape)
```

```
(2712, 12)
(2712, 21)
```

[201]:
```
price_group_price.head()
```

[201]:

```
   price_group_price_id price_group_id added_by_user_id  \
0                     1              1              608
1                     2              2              608
2                     3              1              608
3                     4              3              610
4                     5              4              610

    added_by_backend_user_id       day  price  published  \
0                        NaN  20200312   1100          1
1                        NaN  20200312   1000          1
2                        NaN  20200312   1150          1
3                        NaN  20200312   1400          1
4                        NaN  20200312   1300          1

        client_time_stamp             created_at             updated_at  …  \
0  2020-03-11 19:08:39.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
1  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
2  2020-03-11 19:10:02.0  2020-03-11 19:10:05.0  2020-03-13 07:09:51.0  …
3  2020-03-11 20:09:17.0  2020-03-11 20:09:27.0  2020-03-15 20:41:01.0  …
4  2020-03-11 20:11:05.0  2020-03-11 20:11:11.0  2020-03-15 20:41:01.0  …

          date Unnamed: 0  user_id              buyer      buyer_receipt  \
0  2020-03-12      116.0      608  mill//8//193//nan  mill//8//nan//nan
1  2020-03-12      116.0      608  mill//8//193//nan  mill//8//nan//nan
2  2020-03-12      116.0      608  mill//8//193//nan  mill//8//nan//nan
3  2020-03-12       34.0      610   lr//nan//nan//54   lr//nan//nan//54
4  2020-03-12       34.0      610   lr//nan//nan//54   lr//nan//nan//54

           lng       lat     buyer_kecamatan_desa counterfactual_buyer  \
0  102.890497 -0.602906     KEMPAS__HARAPAN TANI    mill//8//193//nan
1  102.890497 -0.602906     KEMPAS__HARAPAN TANI    mill//8//193//nan
2  102.890497 -0.602906     KEMPAS__HARAPAN TANI    mill//8//193//nan
3  102.508265 -0.719742  BATANG GANSAL__SEBERIDA     lr//nan//nan//54
4  102.508265 -0.719742  BATANG GANSAL__SEBERIDA     lr//nan//nan//54

  counterfactual_buyer_kecamatan_desa
0                KEMPAS__HARAPAN TANI
1                KEMPAS__HARAPAN TANI
2                KEMPAS__HARAPAN TANI
3             BATANG GANSAL__SEBERIDA
4             BATANG GANSAL__SEBERIDA

[5 rows x 21 columns]
```

[202]:
```
price_group_price.shape
```

[202]: (2712, 21)

[203]:
```
special_price_daily = price_group_price.
↪drop_duplicates(subset=['price_group_id','added_by_user_id','date'],
                                                        keep='last'
                                                       )

special_price_daily.shape
```

[203]: (2617, 21)

[204]:
```
special_price_daily.drop(columns=['user_id'], inplace = True)
```

[205]:
```
print(special_price_daily.shape)

cols = ['price_group_id','added_by_user_id','user_id','date_added']

special_price_daily = special_price_daily.merge(price_group_user[cols],
                                                on =␣
↪['price_group_id','added_by_user_id'],
                                                 how='outer'
                                                )

print(special_price_daily.shape)
```

```
(2617, 20)
(52868, 22)
```

[206]:
```
special_price_daily['posted_before_join'] = special_price_daily['date'] <␣
↪special_price_daily['date_added']
```

[207]:
```
special_price_daily['posted_before_join'].value_counts()
```

[207]:
```
False    49448
True      3420
Name: posted_before_join, dtype: int64
```

[208]:
```
special_price_daily['source'] ='special price membership'
```

[209]:
```
mask = ~special_price_daily['posted_before_join']

special_price_daily = special_price_daily[mask]
```

[210]:
```
'''
def special_price_case(row):
    print(row)
    buyer = transactions_sql['buyer'].loc[row]
```

```
    user_id = transactions_sql['user_id'].loc[row]
    date = transactions_sql['date'].loc[row]

    mask = ((special_price_daily['buyer']==buyer) &
            (special_price_daily['user_id']==user_id) &
            (special_price_daily['date']==date)
           )

    tmp = special_price_daily[mask]

    output = np.nan

    if tmp.empty:
        output = False
    else :
        output = True

    return output
'''
```

[210]:
```
"\ndef special_price_case(row):\n    print(row)\n    buyer =
transactions_sql['buyer'].loc[row]\n    user_id =
transactions_sql['user_id'].loc[row]\n    date =
transactions_sql['date'].loc[row]\n    \n    mask =
((special_price_daily['buyer']==buyer) &\n
(special_price_daily['user_id']==user_id) &\n
(special_price_daily['date']==date)\n           )\n    \n    tmp =
special_price_daily[mask]\n    \n    output = np.nan \n    \n    if tmp.empty:\n
output = False\n    else :\n        output = True\n        \n    return
output\n"
```

[211]:
```
#special_price_case(0)
```

[212]:
```
#special_price_daily.reset_index(inplace=True)

special_price_daily['special_price_daily_index'] = special_price_daily.index

special_price_daily.head()
```

[212]:
```
  price_group_price_id price_group_id added_by_user_id  \
0                    2              2              608
1                    2              2              608
2                    2              2              608
3                    2              2              608
4                    7              2              608

   added_by_backend_user_id         day   price  published  \
```

```
0                      NaN  20200312.0  1000.0        1.0
1                      NaN  20200312.0  1000.0        1.0
2                      NaN  20200312.0  1000.0        1.0
3                      NaN  20200312.0  1000.0        1.0
4                      NaN  20200313.0  1000.0        1.0

       client_time_stamp             created_at             updated_at  …  \
0  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
1  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
2  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
3  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  2020-03-13 07:09:51.0  …
4  2020-03-12 21:01:29.0  2020-03-12 21:01:44.0  2020-03-13 07:09:51.0  …

          lng       lat  buyer_kecamatan_desa counterfactual_buyer  \
0  102.890497 -0.602906  KEMPAS__HARAPAN TANI    mill//8//193//nan
1  102.890497 -0.602906  KEMPAS__HARAPAN TANI    mill//8//193//nan
2  102.890497 -0.602906  KEMPAS__HARAPAN TANI    mill//8//193//nan
3  102.890497 -0.602906  KEMPAS__HARAPAN TANI    mill//8//193//nan
4  102.890497 -0.602906  KEMPAS__HARAPAN TANI    mill//8//193//nan

  counterfactual_buyer_kecamatan_desa  user_id  date_added posted_before_join  \
0                KEMPAS__HARAPAN TANI      609  2020-03-11              False
1                KEMPAS__HARAPAN TANI      310  2020-03-11              False
2                KEMPAS__HARAPAN TANI      200  2020-03-11              False
3                KEMPAS__HARAPAN TANI       97  2020-03-11              False
4                KEMPAS__HARAPAN TANI      609  2020-03-11              False

                    source special_price_daily_index
0  special price membership                        0
1  special price membership                        1
2  special price membership                        2
3  special price membership                        3
4  special price membership                        4

[5 rows x 25 columns]
```

[213]:
```
cols = ['user_id','date','buyer','special_price_daily_index']

special_price_daily[cols].drop_duplicates(subset=['user_id','date','buyer']).
↪dropna()
```

[213]:
```
   user_id        date              buyer  special_price_daily_index
0      609  2020-03-12  mill//8//193//nan                          0
1      310  2020-03-12  mill//8//193//nan                          1
2      200  2020-03-12  mill//8//193//nan                          2
3       97  2020-03-12  mill//8//193//nan                          3
4      609  2020-03-13  mill//8//193//nan                          4
```

```
…       …         …                    …                                  …
52736    2194  2021-02-04  lr//nan//nan//121                              52736
52737    2195  2021-02-04  lr//nan//nan//121                              52737
52738    2196  2021-02-04  lr//nan//nan//121                              52738
52739    1702  2021-02-04  lr//nan//nan//121                              52739
52740    2197  2021-02-04  lr//nan//nan//121                              52740

[47775 rows x 4 columns]
```

[214]:
```
special_price_daily['user_id'] = special_price_daily['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[215]:
```
print(transactions_sql.shape)

cols = ['user_id','date','buyer','special_price_daily_index']

mergee = special_price_daily[cols].
↪drop_duplicates(subset=['user_id','date','buyer']).dropna()

transactions_sql = transactions_sql.merge(mergee, on =␣
↪['user_id','date','buyer'], how = 'left')

print(transactions_sql.shape)
```

```
(88253, 78)
(88253, 79)
```

[216]:
```
transactions_sql['special_price_daily_index'].notna().sum()
```

[216]:
```
674
```

[217]:
```
transactions_sql['special_price_case'] =␣
↪transactions_sql['special_price_daily_index'].notna()
```

[218]:
```
transactions_sql['special_price_case'].value_counts()
```

[218]:
```
False    87579
True       674
Name: special_price_case, dtype: int64
```

3.6 Create counterfactual_price_buyer_receipt

[219]:
```
buyers_receipt_list = list(do_ramp_visit['buyer_receipt'].dropna().unique())
```

[220]:
```
startdate = pd.to_datetime("2018-12-01")
enddate = transactions_sql['date'].max()
```

```
counterfactual_price_buyer_receipt = pd.DataFrame(index= pd.
 ↪date_range(start=startdate, end=enddate),
                                    columns = buyers_receipt_list)

counterfactual_price_buyer_receipt.reset_index(inplace=True)#.melt()

counterfactual_price_buyer_receipt = counterfactual_price_buyer_receipt.
 ↪melt(id_vars=['index'])

counterfactual_price_buyer_receipt.rename(columns= {'index':'date',
                                     'variable':'buyer_receipt'
                                    },
                            inplace=True
                           )

counterfactual_price_buyer_receipt.drop(['value'], axis=1,inplace=True)

counterfactual_price_buyer_receipt['date'] = pd.
 ↪to_datetime(counterfactual_price_buyer_receipt['date']).dt.date

counterfactual_price_buyer_receipt.head()
```

[220]:

	date	buyer_receipt
0	2018-12-01	lr//nan//nan//11
1	2018-12-02	lr//nan//nan//11
2	2018-12-03	lr//nan//nan//11
3	2018-12-04	lr//nan//nan//11
4	2018-12-05	lr//nan//nan//11

[221]:
```
print(fruit_sale.shape)

cols = ['user_id','date','buyer','special_price_daily_index']

mergee = special_price_daily[cols].
 ↪drop_duplicates(subset=['user_id','date','buyer']).dropna()

fruit_sale= fruit_sale.merge(mergee, on = ['user_id','date','buyer'], how =␣
 ↪'left')

print(fruit_sale.shape)
```

```
(5619, 32)
(5619, 33)
```

[222]:
```
fruit_sale['special_price_daily_index'].value_counts()
```

[222]:
```
35127.0    4
35355.0    4
37580.0    3
18010.0    3
34140.0    3
           ..
39413.0    1
34019.0    1
52337.0    1
34757.0    1
35214.0    1
Name: special_price_daily_index, Length: 131, dtype: int64
```

[223]:
```
cols =␣
 ↪['date','buyer','buyer_receipt','price_fruit_sale','grade_fruit_sale','price_id']

mask = fruit_sale['special_price_daily_index'].isna()

fruit_sale_prices = fruit_sale[cols][mask].dropna()

fruit_sale_prices.rename(columns={'price_fruit_sale':'price_cleaned',
                                  'grade_fruit_sale':'grade_cleaned'},
                         inplace = True
                        )

fruit_sale_prices['source'] = 'fruit_sale'

fruit_sale_prices
```

[223]:

	date	buyer	buyer_receipt	price_cleaned	\
2	2020-10-30	mill//92//203//nan	mill//92//nan//nan	1900.0	
3	2020-10-30	mill//2//15//nan	mill//2//nan//nan	1800.0	
4	2020-10-30	mill//1//nan//nan	mill//1//nan//nan	1850.0	
5	2020-10-30	mill//2//15//nan	mill//2//nan//nan	1500.0	
6	2020-10-30	mill//7//46//nan	mill//7//nan//nan	1500.0	
…	…	…	…	…	
5607	2021-02-14	lr//nan//nan//50	lr//nan//nan//50	1800.0	
5608	2021-02-14	lr//nan//nan//50	lr//nan//nan//50	1800.0	
5609	2021-02-15	lr//nan//nan//51	lr//nan//nan//51	1975.0	
5610	2021-02-15	mill//1//nan//nan	mill//1//nan//nan	1950.0	
5615	2021-02-19	mill//17//45//nan	mill//17//nan//nan	1990.0	

	grade_cleaned	price_id	source
2	4.44	fs_price_id_601	fruit_sale
3	6.00	fs_price_id_602	fruit_sale
4	5.00	fs_price_id_604	fruit_sale
5	5.00	fs_price_id_605	fruit_sale

```
6              4.00    fs_price_id_606  fruit_sale
…              …                …         …
5607           4.32  fs_price_id_86798  fruit_sale
5608           5.03  fs_price_id_86799  fruit_sale
5609           3.98  fs_price_id_86801  fruit_sale
5610           5.01  fs_price_id_86802  fruit_sale
5615           4.07  fs_price_id_86810  fruit_sale

[3205 rows x 7 columns]
```

[224]:
```
cols =␣
↪['date','buyer','buyer_receipt','price_cleaned','grade_cleaned','price_id']

mask = ~(transactions_sql['id'].isin(outlier_id.keys())&
         (~transactions_sql['special_price_case'])
        )

transactions_sql_prices = transactions_sql[cols][mask]

transactions_sql_prices['source'] = 'transaction_sql'

transactions_sql_prices
```

[224]:
```
            date             buyer      buyer_receipt  price_cleaned  \
0     2018-12-02  mill//6//13//nan  mill//6//nan//nan          975.0
1     2018-12-03  mill//2//15//nan  mill//2//nan//nan          920.0
2     2018-12-03  mill//6//13//nan  mill//6//nan//nan          990.0
3     2018-12-03  mill//5//14//nan  mill//5//nan//nan          960.0
4     2018-12-03  mill//3//16//nan  mill//3//nan//nan         1065.0
…            …                …                 …            …
88248 2021-02-09   lr//nan//nan//3   lr//nan//nan//3         1830.0
88249 2021-02-09   lr//nan//nan//4   lr//nan//nan//4            NaN
88250 2021-02-09   lr//nan//nan//4   lr//nan//nan//4            NaN
88251 2021-02-09   lr//nan//nan//4   lr//nan//nan//4         1900.0
88252 2021-02-09   lr//nan//nan//4   lr//nan//nan//4         1900.0

      grade_cleaned                    price_id           source
0               NaN    transactions_price_id_49  transaction_sql
1               NaN    transactions_price_id_50  transaction_sql
2               NaN    transactions_price_id_51  transaction_sql
3               NaN    transactions_price_id_52  transaction_sql
4               NaN    transactions_price_id_53  transaction_sql
…               …                           …                …
88248           NaN  transactions_price_id_89203  transaction_sql
88249           NaN  transactions_price_id_89204  transaction_sql
88250           NaN  transactions_price_id_89205  transaction_sql
88251           NaN  transactions_price_id_89206  transaction_sql
```

```
88252            NaN  transactions_price_id_89207  transaction_sql

[88091 rows x 7 columns]
```

[225]:
```
users_receipt_duplicates['duplicates_final'].value_counts()
```

[225]:
```
Not Duplicate             323
Duplicate                 185
Not DuplicateDuplicate      3
Not a receipt               3
Name: duplicates_final, dtype: int64
```

[226]:
```
print(users_receipt_duplicates.shape)

cols = ['date','buyer_receipt','special_price_daily_index']

mergee = special_price_daily[cols].
 ↪drop_duplicates(subset=['date','buyer_receipt']).dropna()

users_receipt_duplicates= users_receipt_duplicates.merge(mergee, on =␣
 ↪['date','buyer_receipt'], how = 'left')

print(users_receipt_duplicates.shape)
```

```
(5210, 9)
(5210, 10)
```

[227]:
```
cols =␣
 ↪['date','buyer_receipt','imputed_price_manual','imputed_grade_manual','price_id']

mask = users_receipt_duplicates['special_price_daily_index'].isna()

users_receipt_duplicates_prices = users_receipt_duplicates[cols][mask].
 ↪drop_duplicates()

users_receipt_duplicates_prices['source'] = 'users_receipt_duplicates'

users_receipt_duplicates_prices['buyer'] = np.nan

users_receipt_duplicates_prices.rename(columns={'imputed_price_manual':
 ↪'price_cleaned',
                                                 'imputed_grade_manual':
 ↪'grade_cleaned'},
                                        inplace = True
                                       )

users_receipt_duplicates_prices
```

[227]:
```
            date       buyer_receipt  price_cleaned  grade_cleaned  \
4     2020-11-27   lr//nan//nan//113            NaN            NaN
5     2020-11-26    lr//nan//nan//82            NaN            NaN
6     2020-11-19    lr//nan//nan//79         1800.0       3.984674
7     2020-11-19    lr//nan//nan//79         1800.0       4.031008
8     2020-11-19  mill//76//nan//nan            NaN            NaN
…            …                   …              …              …
5196  2018-12-05   mill//2//nan//nan          940.0            NaN
5198  2020-12-13    lr//nan//nan//73         1850.0            NaN
5199  2020-12-13    lr//nan//nan//73         1850.0            NaN
5202  2020-12-15   lr//nan//nan//113         1890.0       4.000000
5203  2020-12-17   lr//nan//nan//113         1890.0       3.977273

                   price_id                      source  buyer
4     manual_price_id_30678  users_receipt_duplicates    NaN
5     manual_price_id_30677  users_receipt_duplicates    NaN
6     manual_price_id_30660  users_receipt_duplicates    NaN
7     manual_price_id_30658  users_receipt_duplicates    NaN
8     manual_price_id_30656  users_receipt_duplicates    NaN
…                       …                        …    …
5196     manual_price_id_10  users_receipt_duplicates    NaN
5198  manual_price_id_30698  users_receipt_duplicates    NaN
5199  manual_price_id_30700  users_receipt_duplicates    NaN
5202  manual_price_id_30707  users_receipt_duplicates    NaN
5203  manual_price_id_30708  users_receipt_duplicates    NaN

[4532 rows x 7 columns]
```

3.6.1 Creating and cleaning price_grade_table

[228]:
```
price_grade_table = pd.
↪concat([transactions_sql_prices,fruit_sale_prices,users_receipt_duplicates_prices
                            ], ignore_index=True)
```

[229]:
```
daily_median_price_unfiltered = pd.DataFrame(price_grade_table.
↪groupby(['date'])['price_cleaned'].mean())

daily_median_price_unfiltered.columns = ['mean_daily_price']

daily_median_price_unfiltered.reset_index(inplace = True)

daily_median_price_unfiltered
```

[229]:
```
         date  mean_daily_price
0  2016-01-01       1555.000000
1  2018-12-02        975.000000
2  2018-12-03        983.750000
```

```
3    2018-12-04        992.500000
4    2018-12-05        986.000000
..          …               …
802  2021-02-10       1905.555556
803  2021-02-13       1894.000000
804  2021-02-14       1800.000000
805  2021-02-15       1962.500000
806  2021-02-19       1990.000000

[807 rows x 2 columns]
```

[230]:
```
print(price_grade_table.shape)

price_grade_table = price_grade_table.merge(daily_median_price_unfiltered,
                                            on='date',
                                            how = 'left'
                                           )

print(price_grade_table.shape)
```

```
(95828, 7)
(95828, 8)
```

[231]:
```
def get_prev_day_price(data, row):
    date = data['date'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    prev_date = date - timedelta(days=1)
    mask = ((data['date']==prev_date) &
            (data['buyer_receipt']==buyer_receipt)
           )

    output = data['price_cleaned'][mask].median()

    return output
```

[232]:
```
def get_next_day_price(data, row):
    date = data['date'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    next_date = date + timedelta(days=1)
    mask = ((data['date']==next_date) &
            (data['buyer_receipt']==buyer_receipt)
           )

    output = data['price_cleaned'][mask].median()
```

```
    return output
```

[233]:
```
price_grade_table['prev_day_price'] = [get_prev_day_price(price_grade_table,␣
↪row)
                                       for row in price_grade_table.index]
```

[234]:
```
price_grade_table['next_day_price'] = [get_next_day_price(price_grade_table,␣
↪row)
                                       for row in price_grade_table.index]
```

[235]:
```
price_grade_table['prev_day_price_diff'] = np.
↪abs(price_grade_table['price_cleaned']-price_grade_table['prev_day_price'])
```

[236]:
```
price_grade_table['prev_day_price_diff'].describe()
```

[236]:
```
count    30007.000000
mean        16.397091
std         34.217748
min          0.000000
25%          0.000000
50%          5.000000
75%         20.000000
max       1881.000000
Name: prev_day_price_diff, dtype: float64
```

[237]:
```
mask = price_grade_table['prev_day_price_diff']>500

price_grade_table[mask].shape
```

[237]:
```
(7, 11)
```

[238]:
```
price_grade_table['next_day_price_diff'] = np.
↪abs(price_grade_table['price_cleaned']-price_grade_table['next_day_price'])
```

[239]:
```
price_grade_table['mean_price_diff'] = np.
↪abs(price_grade_table['price_cleaned']-price_grade_table['mean_daily_price'])
```

[240]:
```
price_grade_table['mean_price_diff'].describe()
```

[240]:
```
count    35584.000000
mean        57.134699
std         55.194095
min          0.000000
25%         21.792453
50%         45.731756
75%         74.662162
max       1852.121212
```

```
Name: mean_price_diff, dtype: float64
```

[241]:
```
mask = price_grade_table['mean_price_diff']>400

price_grade_table[mask].shape
```

[241]:
```
(44, 13)
```

[242]:
```
price_grade_table.head()
```

[242]:

	date	buyer	buyer_receipt	price_cleaned	grade_cleaned	price_id	source	mean_daily_price	prev_day_price	next_day_price	prev_day_price_diff	next_day_price_diff	mean_price_diff
0	2018-12-02	mill//6//13//nan	mill//6//nan//nan	975.0	NaN	transactions_price_id_49	transaction_sql	975.00	NaN	990.0	NaN	15.0	0.00
1	2018-12-03	mill//2//15//nan	mill//2//nan//nan	920.0	NaN	transactions_price_id_50	transaction_sql	983.75	NaN	920.0	NaN	0.0	63.75
2	2018-12-03	mill//6//13//nan	mill//6//nan//nan	990.0	NaN	transactions_price_id_51	transaction_sql	983.75	975.0	NaN	15.0	NaN	6.25
3	2018-12-03	mill//5//14//nan	mill//5//nan//nan	960.0	NaN	transactions_price_id_52	transaction_sql	983.75	NaN	NaN	NaN	NaN	23.75
4	2018-12-03	mill//3//16//nan	mill//3//nan//nan	1065.0	NaN	transactions_price_id_53	transaction_sql	983.75	NaN	1065.0	NaN	0.0	81.25

[243]:
```
outlier_price_id = []

mask = ((price_grade_table['mean_price_diff']>400) |
        (price_grade_table['prev_day_price_diff']>500) |
        (price_grade_table['next_day_price_diff']>500)
       )

for price_id in price_grade_table['price_id'][mask]:
    outlier_price_id.append(price_id)
```

[244]:
```
len(outlier_price_id)
```

[244]:
```
47
```

[ ]:
```

```

[245]:
```
price_grade_table.shape

mask = ~price_grade_table['price_id'].isin(outlier_price_id)

price_grade_table = price_grade_table[mask].reset_index()

price_grade_table.shape
```

[245]:
```
(95781, 14)
```

[246]:
```
price_grade_table.tail()
```

[246]:

	index	date	buyer	buyer_receipt	price_cleaned	grade_cleaned	price_id	source	mean_daily_price	prev_day_price	next_day_price	prev_day_price_diff	next_day_price_diff	mean_price_diff
95776	95823	2018-12-05	NaN	mill//2//nan//nan	940.0	NaN	manual_price_id_10	users_receipt_duplicates	986.000000	920.0	940.0	20.0	0.0	46.000000
95777	95824	2020-12-13	NaN	lr//nan//nan//73	1850.0	NaN	manual_price_id_30698	users_receipt_duplicates	1907.560976	NaN	1850.0	NaN	0.0	57.560976
95778	95825	2020-12-13	NaN	lr//nan//nan//73	1850.0	NaN	manual_price_id_30700	users_receipt_duplicates	1907.560976	NaN	1850.0	NaN	0.0	57.560976
95779	95826	2020-12-15	NaN	lr//nan//nan//113	1890.0	4.000000	manual_price_id_30707	users_receipt_duplicates	1910.298507	1890.0	NaN	0.0	NaN	20.298507
95780	95827	2020-12-17	NaN	lr//nan//nan//113	1890.0	3.977273	manual_price_id_30708	users_receipt_duplicates	1912.333333	NaN	1890.0	NaN	0.0	22.333333

[247]:
```
print(fruit_sale.shape)

mask = ~fruit_sale['price_id'].isin(outlier_price_id)

fruit_sale = fruit_sale[mask].reset_index()

fruit_sale.shape
```

```
(5619, 33)
```

[247]: (5609, 34)

[248]:
```
print(users_receipt_duplicates.shape)

mask = ~users_receipt_duplicates['price_id'].isin(outlier_price_id)

users_receipt_duplicates = users_receipt_duplicates[mask].reset_index()

users_receipt_duplicates.shape
```

```
(5210, 10)
```

[248]: (5210, 11)

[249]:
```
print(transactions_sql.shape)

mask = ((~transactions_sql['price_id'].isin(outlier_price_id)) |
        (~transactions_sql['price_id'].isin(outlier_id.keys()))
       )

transactions_sql = transactions_sql[mask].reset_index()

transactions_sql.shape
```

```
(88253, 80)
```

[249]: (88253, 81)

[250]:
```
print(users_receipt_duplicates.shape)

mask = ~users_receipt_duplicates['price_id'].isin(outlier_price_id)

users_receipt_duplicates = users_receipt_duplicates[mask].reset_index()

users_receipt_duplicates.shape
```

```
(5210, 11)
```

[250]: (5210, 12)

[251]:
```
median_price_buyer_receipt = pd.pivot_table(price_grade_table,
                                            index=['date','buyer_receipt'],
                                            values =␣
 ↪['price_cleaned','grade_cleaned'],
                                            aggfunc= np.nanmedian
                                           )

median_price_buyer_receipt.columns = [column + '_buyer_receipt' for column in␣
 ↪median_price_buyer_receipt.columns]

median_price_buyer_receipt.reset_index(inplace=True)

median_price_buyer_receipt['date'] = pd.
 ↪to_datetime(median_price_buyer_receipt['date']).dt.date

median_price_buyer_receipt.head()
```

[251]:

	date	buyer_receipt	grade_cleaned_buyer_receipt	price_cleaned_buyer_receipt
0	2016-01-01	lr//nan//nan//38	3.99	1555.0
1	2018-12-02	mill//6//nan//nan	NaN	975.0
2	2018-12-03	mill//2//nan//nan	NaN	920.0
3	2018-12-03	mill//3//nan//nan	NaN	1065.0
4	2018-12-03	mill//5//nan//nan	NaN	960.0

[252]:
```
max_price_buyer_receipt = pd.pivot_table(price_grade_table,
                                         index=['date','buyer_receipt'],
                                         values = ['price_cleaned'],
                                         aggfunc= np.nanmax
                                         )

max_price_buyer_receipt.columns = ['max_' + column + '_buyer_receipt' for␣
 ↪column in max_price_buyer_receipt.columns]

max_price_buyer_receipt.reset_index(inplace=True)

max_price_buyer_receipt['date'] = pd.
 ↪to_datetime(max_price_buyer_receipt['date']).dt.date
```

```
max_price_buyer_receipt.head()
```

[252]:

	date	buyer_receipt	max_price_cleaned_buyer_receipt
0	2016-01-01	lr//nan//nan//38	1555.0
1	2018-12-02	mill//6//nan//nan	975.0
2	2018-12-03	mill//2//nan//nan	920.0
3	2018-12-03	mill//3//nan//nan	1065.0
4	2018-12-03	mill//5//nan//nan	960.0

[253]:
```
min_grade_buyer_receipt = pd.pivot_table(price_grade_table,
                                         index=['date','buyer_receipt'],
                                         values = ['grade_cleaned'],
                                         aggfunc= np.nanmin
                                         )

min_grade_buyer_receipt.columns = ['min_' + column + '_buyer_receipt' for␣
 ↪column in min_grade_buyer_receipt.columns]

min_grade_buyer_receipt.reset_index(inplace=True)

min_grade_buyer_receipt['date'] = pd.
 ↪to_datetime(min_grade_buyer_receipt['date']).dt.date

min_grade_buyer_receipt.head()
```

[253]:

	date	buyer_receipt	min_grade_cleaned_buyer_receipt
0	2016-01-01	lr//nan//nan//38	3.99
1	2018-12-06	mill//2//nan//nan	4.00
2	2018-12-06	mill//6//nan//nan	3.50
3	2018-12-07	mill//1//nan//nan	4.50
4	2018-12-07	mill//3//nan//nan	8.08

[254]:
```
print(counterfactual_price_buyer_receipt.shape)

counterfactual_price_buyer_receipt = counterfactual_price_buyer_receipt.
 ↪merge(median_price_buyer_receipt,
                                                                        ␣
 ↪on =['date','buyer_receipt'],
                                                                        ␣
 ↪how='left'
                                                                        )

print(counterfactual_price_buyer_receipt.shape)
```

```
(81906, 2)
(81906, 4)
```

[255]:
```
print(counterfactual_price_buyer_receipt.shape)

counterfactual_price_buyer_receipt = counterfactual_price_buyer_receipt.
↪merge(max_price_buyer_receipt,
                                                                        ␣
↪on =['date','buyer_receipt'],
                                                                        ␣
↪how='left'
                                                                        )

print(counterfactual_price_buyer_receipt.shape)
```

```
(81906, 4)
(81906, 5)
```

[256]:
```
print(counterfactual_price_buyer_receipt.shape)

counterfactual_price_buyer_receipt = counterfactual_price_buyer_receipt.
↪merge(min_grade_buyer_receipt,
                                                                        ␣
↪on =['date','buyer_receipt'],
                                                                        ␣
↪how='left'
                                                                        )

print(counterfactual_price_buyer_receipt.shape)
```

```
(81906, 5)
(81906, 6)
```

[257]:
```
counterfactual_price_buyer_receipt.head()
```

[257]:

	date	buyer_receipt	grade_cleaned_buyer_receipt	price_cleaned_buyer_receipt	max_price_cleaned_buyer_receipt	min_grade_cleaned_buyer_receipt
0	2018-12-01	lr//nan//nan//11	NaN	NaN	NaN	
1	2018-12-02	lr//nan//nan//11	NaN	NaN	NaN	
2	2018-12-03	lr//nan//nan//11	NaN	NaN	NaN	
3	2018-12-04	lr//nan//nan//11	NaN	NaN	NaN	
4	2018-12-05	lr//nan//nan//11	NaN	NaN	NaN	

```
0                                   NaN
1                                   NaN
2                                   NaN
3                                   NaN
4                                   NaN
```

3.7 Create connectedness_village_df table

[258]:
```
'''
def create_connectedness_table():
    connectedness_village_df = pd.DataFrame(index =␣
↪relevant_village['kecamatan_desa'],
                                            columns =␣
↪relevant_village['kecamatan_desa'])
    for i in connectedness_village_df.index:
        for j in connectedness_village_df.columns:
            mask_i = relevant_village['kecamatan_desa']==i
            mask_j = relevant_village['kecamatan_desa']==j
            a = relevant_village[mask_i].index.values[0]
            b = relevant_village[mask_j].index.values[0]
            connectedness_village_df[j].loc[i]=(
                relevant_village['geometry'].loc[a].
↪touches(relevant_village['geometry'].loc[b])
            )
    connectedness_village_df.reset_index(inplace = True)

    connectedness_village_df.rename(columns={'kecamatan_desa':␣
↪'buyer_kecamatan_desa'}, inplace=True)

    connectedness_village_df_melted = pd.melt(connectedness_village_df,␣
↪id_vars='buyer_kecamatan_desa',
                                              ␣
↪value_vars=list(connectedness_village_df.columns[1:])
                                              )
    #print(connectedness_village_df_melted.head())
    connectedness_village_df_melted.columns =␣
↪['buyer_kecamatan_desa','kecamatan_desa','kecamatan_desa_connected']

    connectedness_village_df_melted['kecamatan_desa_connected'] =␣
↪connectedness_village_df_melted['kecamatan_desa_connected']*1

    connectedness_village_df_melted['same_village'] =␣
↪connectedness_village_df_melted['kecamatan_desa']==connectedness_village_df_melted['buyer_k

    #mask = (~(connectedness_village_df_melted['same_village']))
```

```
    #connectedness_village_df_melted = connectedness_village_df_melted[mask]

    return connectedness_village_df_melted
'''
```

[258]:
```
"\ndef create_connectedness_table():\n    connectedness_village_df =
pd.DataFrame(index = relevant_village['kecamatan_desa'], \n
columns = relevant_village['kecamatan_desa'])\n    for i in
connectedness_village_df.index:\n        for j in
connectedness_village_df.columns:\n            mask_i =
relevant_village['kecamatan_desa']==i\n            mask_j =
relevant_village['kecamatan_desa']==j\n            a =
relevant_village[mask_i].index.values[0]\n            b =
relevant_village[mask_j].index.values[0]\n
connectedness_village_df[j].loc[i]=(\n                relevant_village['geometry
'].loc[a].touches(relevant_village['geometry'].loc[b])\n            )\n
connectedness_village_df.reset_index(inplace = True)\n\n
connectedness_village_df.rename(columns={'kecamatan_desa':
'buyer_kecamatan_desa'}, inplace=True)\n\n    connectedness_village_df_melted =
pd.melt(connectedness_village_df, id_vars='buyer_kecamatan_desa',\n
value_vars=list(connectedness_village_df.columns[1:])\n
) \n    #print(connectedness_village_df_melted.head())\n
connectedness_village_df_melted.columns =
['buyer_kecamatan_desa','kecamatan_desa','kecamatan_desa_connected']\n\n
connectedness_village_df_melted['kecamatan_desa_connected'] =
connectedness_village_df_melted['kecamatan_desa_connected']*1\n    \n
connectedness_village_df_melted['same_village'] = connectedness_village_df_melte
d['kecamatan_desa']==connectedness_village_df_melted['buyer_kecamatan_desa']\n\n
#mask = (~(connectedness_village_df_melted['same_village']))\n\n
#connectedness_village_df_melted = connectedness_village_df_melted[mask]\n    \n
return connectedness_village_df_melted\n"
```

[259]:
```
'''
connectedness_village_df = create_connectedness_table()
'''
```

[259]:
```
'\nconnectedness_village_df = create_connectedness_table()\n'
```

[260]:
```
'''
mask = connectedness_village_df['kecamatan_desa_connected']==1

village_network = nx.from_pandas_edgelist(connectedness_village_df[mask],
                                          ␣
 ↪'kecamatan_desa','buyer_kecamatan_desa',
                                           create_using=nx.MultiGraph(),
                                          )
'''
```

[260]:
```
"\nmask =
connectedness_village_df['kecamatan_desa_connected']==1\n\nvillage_network =
nx.from_pandas_edgelist(connectedness_village_df[mask], \n
'kecamatan_desa','buyer_kecamatan_desa',\n
create_using=nx.MultiGraph(),\n                                                )\n"
```

[261]:
```
'''
def get_shortest_path_length(data,row):
    source = data['kecamatan_desa'].loc[row]
    target = data['buyer_kecamatan_desa'].loc[row]

    if ((source in list(village_network.nodes())) & (target in␣
↪list(village_network.nodes()))):
        shortest_path = nx.shortest_path(village_network, source=source,
                                         target=target)

        output = len(shortest_path) -1
    else :
        output = np.nan

    return output
'''
```

[261]:
```
"\ndef get_shortest_path_length(data,row):\n    source =
data['kecamatan_desa'].loc[row]\n    target =
data['buyer_kecamatan_desa'].loc[row]\n    \n    if ((source in
list(village_network.nodes())) & (target in list(village_network.nodes()))):\n
shortest_path = nx.shortest_path(village_network, source=source, \n
target=target)\n        \n        output = len(shortest_path) -1\n    else :\n
output = np.nan\n    \n    return output\n"
```

[262]:
```
'''
def get_shortest_path_length(data,row):
    source = data['kecamatan_desa'].loc[row]
    target = data['buyer_kecamatan_desa'].loc[row]

    try:
        shortest_path = nx.shortest_path(village_network, source=source,
                                         target=target)

        output = len(shortest_path) -1
    except :
        output = np.nan

    return output
'''
```

[262]:
```
"\ndef get_shortest_path_length(data,row):\n    source =
data['kecamatan_desa'].loc[row]\n    target =
data['buyer_kecamatan_desa'].loc[row]\n    \n    try:\n        shortest_path =
nx.shortest_path(village_network, source=source, \n
target=target)\n        \n        output = len(shortest_path) -1\n    except :\n
output = np.nan\n    \n    return output\n"
```

[263]:
```
'''
connectedness_village_df['shortest_path_length'] =␣
↪[get_shortest_path_length(connectedness_village_df,row)
                                                    for row in␣
↪connectedness_village_df.index
                                                   ]
'''
```

[263]:
```
"\nconnectedness_village_df['shortest_path_length'] =
[get_shortest_path_length(connectedness_village_df,row) \n
for row in connectedness_village_df.index \n
]\n"
```

[264]:
```
'''
connectedness_village_df['shortest_path_length'].describe()
'''
```

[264]:
```
"\nconnectedness_village_df['shortest_path_length'].describe() \n"
```

3.8 Create village_neighbors_df table

[265]:
```
'''
village_neighbors_df = pd.DataFrame(connectedness_village_df['kecamatan_desa'].
↪drop_duplicates())

village_neighbors_df.reset_index(inplace=True)

village_neighbors_df.drop(columns=['index'], inplace=True)

village_neighbors_df
'''
```

[265]:
```
"\nvillage_neighbors_df = pd.DataFrame(connectedness_village_df['kecamatan_desa'
].drop_duplicates())\n\nvillage_neighbors_df.reset_index(inplace=True)\n\nvillag
e_neighbors_df.drop(columns=['index'], inplace=True)\n\nvillage_neighbors_df\n"
```

[266]:
```
'''
def get_village_neighbors(num_village_threshold, row):
    source = village_neighbors_df['kecamatan_desa'].loc[row]
```

```
    mask = ((connectedness_village_df['shortest_path_length']<=num_village_threshold) &
            (connectedness_village_df['kecamatan_desa']==source)
           )

    output = list(connectedness_village_df['buyer_kecamatan_desa'][mask])

    return output
'''
```

[266]:
```
"\ndef get_village_neighbors(num_village_threshold, row):\n    source = village_neighbors_df['kecamatan_desa'].loc[row]\n    mask = ((connectedness_village_df['shortest_path_length'] <=num_village_threshold) &\n(connectedness_village_df['kecamatan_desa']==source)\n           )\n    \n output = list(connectedness_village_df['buyer_kecamatan_desa'][mask])\n    \n return output\n"
```

[267]:
```
'''
village_neighbors_df['village_neighbors_<=5'] = [get_village_neighbors(5, row)for
                                                 row in village_neighbors_df.index]
'''
```

[267]:
```
"\nvillage_neighbors_df['village_neighbors_<=5'] = [get_village_neighbors(5, row) for \n                                                 row in village_neighbors_df.index]\n"
```

3.9 Create counterfactual_price table

[268]:
```
buyers_counterfactual_list = list(do_ramp_visit['buyer'].dropna().unique())
```

[269]:
```
startdate = pd.to_datetime("2018-12-01")
enddate = transactions_sql['date'].max()

counterfactual_price = pd.DataFrame(index= pd.date_range(start=startdate,end=enddate),
                                    columns = buyers_counterfactual_list)

counterfactual_price.reset_index(inplace=True)#.melt()

counterfactual_price = counterfactual_price.melt(id_vars=['index'])

counterfactual_price.rename(columns= {'index':'date',
                                      'variable':'buyer'
                                     },
```

```
                            inplace=True
                           )

counterfactual_price.drop(['value'], axis=1,inplace=True)

counterfactual_price['date'] = pd.to_datetime(counterfactual_price['date']).dt.
↪date

counterfactual_price.head()
```

[269]:

	date	buyer
0	2018-12-01	lr//nan//nan//11
1	2018-12-02	lr//nan//nan//11
2	2018-12-03	lr//nan//nan//11
3	2018-12-04	lr//nan//nan//11
4	2018-12-05	lr//nan//nan//11

[270]:
```
print(counterfactual_price.shape)

buyer_buyer_receipt_lookup = buyers[['buyer','buyer_receipt']].drop_duplicates()

counterfactual_price = counterfactual_price.merge(buyer_buyer_receipt_lookup,
                                                  on = 'buyer',
                                                   how= 'left'
                                                  )

print(counterfactual_price.shape)
```

```
(103587, 2)
(103587, 3)
```

[271]:
```
#counterfactual_price['buyer_receipt'].fillna(counterfactual_price['buyer'],␣
↪inplace = True)
```

[272]:
```
median_price = pd.pivot_table(price_grade_table,␣
↪index=['date','buyer_receipt','buyer'],
                              values = ['price_cleaned','grade_cleaned'],
                              aggfunc= np.nanmedian
                             )

median_price.reset_index(inplace=True)

median_price['date'] = pd.to_datetime(median_price['date']).dt.date

median_price.head()
```

[272]:
```
         date      buyer_receipt             buyer  grade_cleaned  \
0  2016-01-01   lr//nan//nan//38  lr//nan//nan//38           3.99
1  2018-12-02  mill//6//nan//nan  mill//6//13//nan            NaN
2  2018-12-03  mill//2//nan//nan  mill//2//15//nan            NaN
3  2018-12-03  mill//3//nan//nan  mill//3//16//nan            NaN
4  2018-12-03  mill//5//nan//nan  mill//5//14//nan            NaN

   price_cleaned
0         1555.0
1          975.0
2          920.0
3         1065.0
4          960.0
```

[273]:
```
max_price = pd.pivot_table(price_grade_table,␣
↪index=['date','buyer_receipt','buyer'],
                           values = ['price_cleaned'],
                           aggfunc= np.nanmax
                           )

max_price.columns = ['max_' + col for col in max_price.columns]

max_price.reset_index(inplace=True)

max_price['date'] = pd.to_datetime(max_price['date']).dt.date

max_price.head()
```

[273]:
```
         date      buyer_receipt             buyer  max_price_cleaned
0  2016-01-01   lr//nan//nan//38  lr//nan//nan//38             1555.0
1  2018-12-02  mill//6//nan//nan  mill//6//13//nan              975.0
2  2018-12-03  mill//2//nan//nan  mill//2//15//nan              920.0
3  2018-12-03  mill//3//nan//nan  mill//3//16//nan             1065.0
4  2018-12-03  mill//5//nan//nan  mill//5//14//nan              960.0
```

[274]:
```
min_grade = pd.pivot_table(price_grade_table,␣
↪index=['date','buyer_receipt','buyer'],
                           values = ['grade_cleaned'],
                           aggfunc= np.nanmin
                           )

min_grade.columns = ['min_' + col for col in min_grade.columns]

min_grade.reset_index(inplace=True)

min_grade['date'] = pd.to_datetime(min_grade['date']).dt.date
```

```
min_grade.head()
```

[274]:

	date	buyer_receipt	buyer	min_grade_cleaned
0	2016-01-01	lr//nan//nan//38	lr//nan//nan//38	3.99
1	2018-12-06	mill//2//nan//nan	mill//2//1//nan	5.01
2	2018-12-06	mill//2//nan//nan	mill//2//15//nan	4.00
3	2018-12-06	mill//2//nan//nan	mill//2//nan//nan	5.00
4	2018-12-06	mill//6//nan//nan	mill//6//nan//nan	3.50

[275]:
```
print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(median_price, on␣
↪=['date','buyer','buyer_receipt'],
                                                  how='left'
                                                 )

print(counterfactual_price.shape)
```

```
(103587, 3)
(103587, 5)
```

[276]:
```
print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(max_price, on␣
↪=['date','buyer','buyer_receipt'],
                                                  how='left'
                                                 )

print(counterfactual_price.shape)
```

```
(103587, 5)
(103587, 6)
```

[277]:
```
print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(min_grade, on␣
↪=['date','buyer','buyer_receipt'],
                                                  how='left'
                                                 )

print(counterfactual_price.shape)
```

```
(103587, 6)
(103587, 7)
```

[278]:
```
print(counterfactual_price.shape)
```

```
counterfactual_price = counterfactual_price.
 ↪merge(counterfactual_price_buyer_receipt,
                                                  on =['date','buyer_receipt',],
                                                  how='left'
                                                 )

print(counterfactual_price.shape)
```

```
(103587, 7)
(103587, 11)
```

[279]:
```
counterfactual_price.columns
```

[279]:
```
Index(['date', 'buyer', 'buyer_receipt', 'grade_cleaned', 'price_cleaned',
       'max_price_cleaned', 'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt'],
      dtype='object')
```

[280]:
```
counterfactual_price_buyer_receipt.columns
```

[280]:
```
Index(['date', 'buyer_receipt', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt'],
      dtype='object')
```

[281]:
```
counterfactual_price['price_counterfactual_table'] =␣
 ↪counterfactual_price['price_cleaned']
```

[282]:
```
counterfactual_price['price_counterfactual_table'].dropna().shape
```

[282]:
```
(17449,)
```

[283]:
```
counterfactual_price['price_counterfactual_table'].
 ↪fillna(counterfactual_price['price_cleaned_buyer_receipt'],
                                                   inplace = True
                                                  )
```

[284]:
```
counterfactual_price['price_counterfactual_table'].dropna().shape
```

[284]:
```
(32246,)
```

[285]:
```
counterfactual_price['grade_counterfactual_table'] =␣
 ↪counterfactual_price['grade_cleaned']
```

[286]:
```
counterfactual_price['grade_counterfactual_table'].dropna().shape
```

[286]: (3624,)

[287]:
```
counterfactual_price['grade_counterfactual_table'].
↪fillna(counterfactual_price['grade_cleaned_buyer_receipt'],
                                                    inplace = True
                                                   )
```

[288]:
```
counterfactual_price['grade_counterfactual_table'].dropna().shape
```

[288]: (15027,)

[289]:
```
counterfactual_price['post_grading_price_counterfactual_table'] =␣
↪(counterfactual_price['price_counterfactual_table']*
                                                                  ␣
↪(1-counterfactual_price['grade_counterfactual_table']/100)
                                                                  )
```

[290]:
```
counterfactual_price['max_price_counterfactual_table'] =␣
↪counterfactual_price['max_price_cleaned']
```

[291]:
```
counterfactual_price['max_price_counterfactual_table'].
↪fillna(counterfactual_price['max_price_cleaned_buyer_receipt'],
                                                      inplace = True
                                                     )
```

[292]:
```
counterfactual_price['max_price_counterfactual_table'].dropna().shape
```

[292]: (32246,)

[293]:
```
counterfactual_price
```

[293]:

	date	buyer	buyer_receipt	grade_cleaned	\
0	2018-12-01	lr//nan//nan//11	NaN	NaN	
1	2018-12-02	lr//nan//nan//11	NaN	NaN	
2	2018-12-03	lr//nan//nan//11	NaN	NaN	
3	2018-12-04	lr//nan//nan//11	NaN	NaN	
4	2018-12-05	lr//nan//nan//11	NaN	NaN	
…	…	…	…	…	
103582	2021-02-06	mill//105//208//nan	mill//105//nan//nan	NaN	
103583	2021-02-07	mill//105//208//nan	mill//105//nan//nan	NaN	
103584	2021-02-08	mill//105//208//nan	mill//105//nan//nan	NaN	
103585	2021-02-09	mill//105//208//nan	mill//105//nan//nan	NaN	
103586	2021-02-10	mill//105//208//nan	mill//105//nan//nan	NaN	

	price_cleaned	max_price_cleaned	min_grade_cleaned	\
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	

```
2                NaN                NaN                NaN
3                NaN                NaN                NaN
4                NaN                NaN                NaN
…                …                  …                  …
103582           NaN                NaN                NaN
103583           NaN                NaN                NaN
103584           NaN                NaN                NaN
103585           NaN                NaN                NaN
103586           NaN                NaN                NaN

        grade_cleaned_buyer_receipt  price_cleaned_buyer_receipt  \
0                               NaN                          NaN
1                               NaN                          NaN
2                               NaN                          NaN
3                               NaN                          NaN
4                               NaN                          NaN
…                               …                            …
103582                          NaN                          NaN
103583                          NaN                          NaN
103584                          NaN                          NaN
103585                          NaN                          NaN
103586                          NaN                          NaN

        max_price_cleaned_buyer_receipt  min_grade_cleaned_buyer_receipt  \
0                                   NaN                              NaN
1                                   NaN                              NaN
2                                   NaN                              NaN
3                                   NaN                              NaN
4                                   NaN                              NaN
…                                   …                                …
103582                              NaN                              NaN
103583                              NaN                              NaN
103584                              NaN                              NaN
103585                              NaN                              NaN
103586                              NaN                              NaN

        price_counterfactual_table  grade_counterfactual_table  \
0                              NaN                         NaN
1                              NaN                         NaN
2                              NaN                         NaN
3                              NaN                         NaN
4                              NaN                         NaN
…                              …                           …
103582                         NaN                         NaN
103583                         NaN                         NaN
103584                         NaN                         NaN
103585                         NaN                         NaN
```

```
103586                          NaN                          NaN

        post_grading_price_counterfactual_table  \
0                                           NaN
1                                           NaN
2                                           NaN
3                                           NaN
4                                           NaN
…                                           …
103582                                      NaN
103583                                      NaN
103584                                      NaN
103585                                      NaN
103586                                      NaN

        max_price_counterfactual_table
0                                  NaN
1                                  NaN
2                                  NaN
3                                  NaN
4                                  NaN
…                                  …
103582                             NaN
103583                             NaN
103584                             NaN
103585                             NaN
103586                             NaN

[103587 rows x 15 columns]
```

[294]:
```
counterfactual_price['min_grade_counterfactual_table'] =␣
↪counterfactual_price['min_grade_cleaned']
```

[295]:
```
counterfactual_price['min_grade_counterfactual_table'].
↪fillna(counterfactual_price['min_grade_cleaned_buyer_receipt'],
                                                    inplace = True
                                                   )
```

[296]:
```
counterfactual_price['max_post_grading_price_counterfactual_table'] =␣
↪(counterfactual_price['max_price_counterfactual_table']*

                                                                    ␣
↪(1-counterfactual_price['min_grade_counterfactual_table']/100)
                                                                  )
```

[297]:
```
print(counterfactual_price.shape)

cols = ['buyer','buyer_kecamatan_desa']
```

```
counterfactual_price = counterfactual_price.merge(buyers[cols],
                                                  on = 'buyer',
                                                  how = 'left'
                                                 )

print(counterfactual_price.shape)
```

```
(103587, 17)
(103587, 18)
```

[298]:
```
'''
cols = ['geometry','kecamatan_desa']

print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(relevant_village[cols],
                                                  left_on =␣
 ↪'buyer_kecamatan_desa',
                                                  right_on = 'kecamatan_desa',
                                                  how = 'left'
                                                 )

print(counterfactual_price.shape)
'''
```

[298]:
```
"\ncols = ['geometry','kecamatan_desa']\n\nprint(counterfactual_price.shape)\n\n
counterfactual_price = counterfactual_price.merge(relevant_village[cols],\n
left_on = 'buyer_kecamatan_desa',\n
right_on = 'kecamatan_desa',\n
how = 'left'\n
)\n\nprint(counterfactual_price.shape)\n"
```

3.10 Create do_ramp_visit

[299]:
```
print(do_ramp_visit.shape)

do_ramp_visit = do_ramp_visit.merge(counterfactual_price,
                                    on = ['date','buyer_receipt','buyer'],
                                    how = 'left'
                                   )

print(do_ramp_visit.shape)
```

```
(620433, 25)
(620433, 40)
```

[300]:
```
do_ramp_visit.columns
```

[300]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt', 'grade_cleaned', 'price_cleaned', 'max_price_cleaned',
       'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa'],
      dtype='object')
```

[301]:
```
cols = ['user_id','date', 'village_cleaned', 'kecamatan','kecamatan_desa']

mergee = users[cols]

mergee.rename(columns= {'date':'join_date'},
              inplace =True
             )

print(do_ramp_visit.shape)

do_ramp_visit = do_ramp_visit.merge(mergee, on='user_id',
                                    how='left'
                                    )

print(do_ramp_visit.shape)
```

```
(620433, 40)
(620433, 44)
```

[302]:
```
do_ramp_visit.columns
```

[302]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt', 'grade_cleaned', 'price_cleaned', 'max_price_cleaned',
       'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa',
```

```
       'join_date', 'village_cleaned', 'kecamatan', 'kecamatan_desa'],
      dtype='object')
```

[303]:
```
do_ramp_visit['join_date'] = pd.to_datetime(do_ramp_visit['join_date']).dt.date
```

[304]:
```
do_ramp_visit['membership_length'] =  (do_ramp_visit['date']-␣
↪do_ramp_visit['join_date']).dt.days
```

[305]:
```
do_ramp_visit['membership_length'].describe()
```

[305]:
```
count    620433.000000
mean        190.880461
std         172.991219
min       -1146.000000
25%          40.000000
50%         152.000000
75%         285.000000
max         797.000000
Name: membership_length, dtype: float64
```

[306]:
```
def check_new_buyer(row):
    user_id = do_ramp_visit['user_id'].loc[row]
    mask = users_do_ramp['user_id']==user_id
    tmp_df = users_do_ramp[mask]
    if tmp_df.empty:
        output = np.nan
    else:
        buyer_list = list(tmp_df['buyer'])
        if do_ramp_visit['buyer'].loc[row] in buyer_list:
            output = 0
        else :
            output = 1
    return output
```

[307]:
```
do_ramp_visit['new_buyer'] =  [check_new_buyer(row) for row in do_ramp_visit.
↪index]
```

[308]:
```
connectedness_village_df.shape
```

[308]:
```
(255025, 6)
```

[309]:
```
connectedness_village_df.columns
```

[309]:
```
Index(['buyer_kecamatan_desa', 'kecamatan_desa', 'kecamatan_desa_connected',
       'same_village', 'user_kecamatan_desa', 'shortest_path_length'],
      dtype='object')
```

[310]:
```
#do_ramp_visit.rename(columns={'buyer_kecamatan_desa_y':
↪'buyer_kecamatan_desa'}, inplace=True)
```

[311]:
```
do_ramp_visit.columns
```

[311]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt', 'grade_cleaned', 'price_cleaned', 'max_price_cleaned',
       'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa',
       'join_date', 'village_cleaned', 'kecamatan', 'kecamatan_desa',
       'membership_length', 'new_buyer'],
      dtype='object')
```

[312]:
```
print(do_ramp_visit.shape)

do_ramp_visit = do_ramp_visit.merge(connectedness_village_df,
                                    on =␣
↪['buyer_kecamatan_desa','kecamatan_desa'],
                                    how = 'left'
                                   )

print(do_ramp_visit.shape)
```

```
(620433, 46)
(620433, 50)
```

[313]:
```
do_ramp_visit[['kecamatan_desa','buyer_kecamatan_desa']].dropna()
```

[313]:

	kecamatan_desa	buyer_kecamatan_desa
1	BATANG GANSAL__SEBERIDA	SEBERIDA__PANGKALAN KASAI
2	BATANG GANSAL__SEBERIDA	BATANG CENAKU__BUKIT LIPAI
3	BATANG GANSAL__SEBERIDA	BATANG CENAKU__AUR CINA
4	BATANG GANSAL__SEBERIDA	RENGAT BARAT__PEMATANG REBA
5	BATANG GANSAL__SEBERIDA	SEBERIDA__PANGKALAN KASAI
…	…	…
620422	BATANG GANSAL__SUNGAI AKAR	BATANG GANSAL__RINGIN
620423	BATANG GANSAL__SUNGAI AKAR	KERITANG__PENGALIHAN
620424	BATANG GANSAL__SUNGAI AKAR	KERITANG__PENGALIHAN
620425	BATANG GANSAL__SUNGAI AKAR	PERANAP__SEMELINANG DARAT
620426	BATANG GANSAL__SUNGAI AKAR	PERANAP__SEMELINANG DARAT

```
[600725 rows x 2 columns]
```

[314]:
```
do_ramp_visit.head()
```

[314]:
```
   id user_id do_id           date_time        date    month  week  biweek  \
0  3     124   nan 2019-05-09 03:28:18  2019-05-09  2019-05    19      10
1  5     201   nan 2019-05-09 06:27:29  2019-05-09  2019-05    19      10
2  6     201   nan 2019-05-09 06:27:33  2019-05-09  2019-05    19      10
3  7     201   nan 2019-05-09 06:27:35  2019-05-09  2019-05    19      10
4  8     201   nan 2019-05-09 06:27:39  2019-05-09  2019-05    19      10

   triweek  quadweek  …   join_date village_cleaned      kecamatan  \
0        7         5  …  2019-01-22        Seberida  Batang Gansal
1        7         5  …  2019-04-11        Seberida  Batang Gansal
2        7         5  …  2019-04-11        Seberida  Batang Gansal
3        7         5  …  2019-04-11        Seberida  Batang Gansal
4        7         5  …  2019-04-11        Seberida  Batang Gansal

            kecamatan_desa membership_length new_buyer  \
0  BATANG GANSAL__SEBERIDA               107       1.0
1  BATANG GANSAL__SEBERIDA                28       1.0
2  BATANG GANSAL__SEBERIDA                28       1.0
3  BATANG GANSAL__SEBERIDA                28       1.0
4  BATANG GANSAL__SEBERIDA                28       1.0

  kecamatan_desa_connected same_village      user_kecamatan_desa  \
0                      NaN          NaN                      NaN
1                      0.0        False  BATANG GANSAL__SEBERIDA
2                      0.0        False  BATANG GANSAL__SEBERIDA
3                      0.0        False  BATANG GANSAL__SEBERIDA
4                      0.0        False  BATANG GANSAL__SEBERIDA

  shortest_path_length
0                  NaN
1                  3.0
2                  3.0
3                  2.0
4                  4.0

[5 rows x 50 columns]
```

[315]:
```
do_ramp_visit['shortest_path_length'].describe()
```

[315]:
```
count    575819.000000
mean          3.066543
std           2.240853
```

```
min            0.000000
25%            1.000000
50%            3.000000
75%            4.000000
max           14.000000
Name: shortest_path_length, dtype: float64
```

[316]:
```
np.quantile(do_ramp_visit['shortest_path_length'].dropna(),0.8)
```

[316]:
```
5.0
```

[317]:
```
np.quantile(do_ramp_visit['shortest_path_length'].dropna(),0.9)
```

[317]:
```
6.0
```

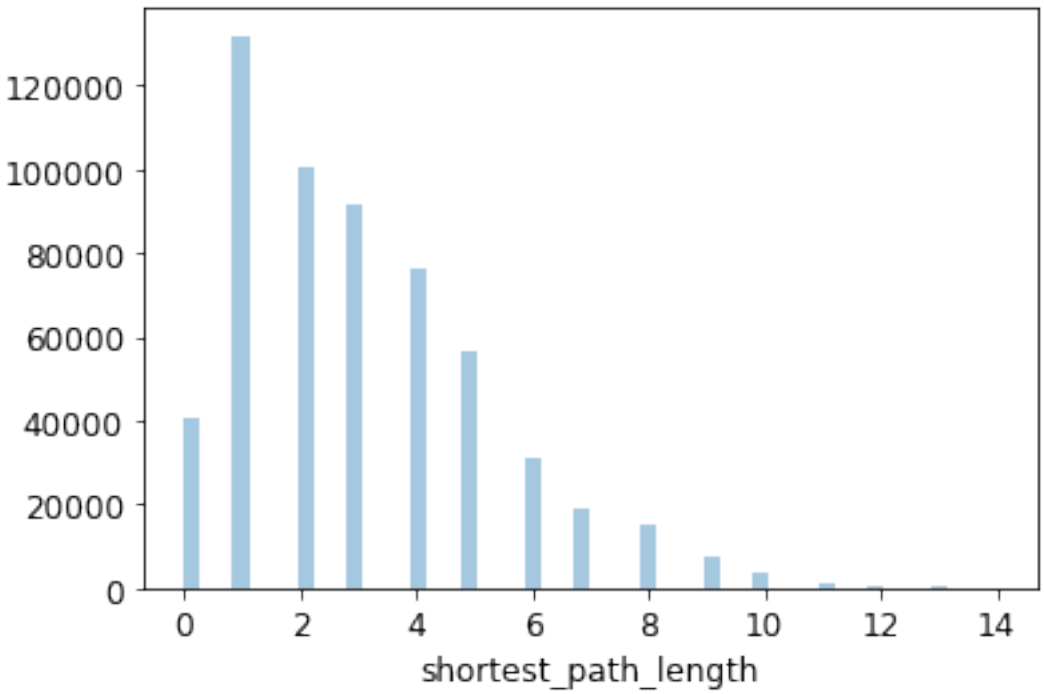
[318]:
```
np.quantile(do_ramp_visit['shortest_path_length'].dropna(),0.95)
```

[318]:
```
7.0
```

[319]:
```
sns.distplot(do_ramp_visit['shortest_path_length'].dropna(), kde=False)
```

```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[319]:
```
<AxesSubplot:xlabel='shortest_path_length'>
```



3.11 Create do_ramp_visit_daily

[320]:
```
users_analyzed = list(last_most_transaction_do_ramp['user_id'].unique())

len(users_analyzed)
```

[320]:
```
967
```

[321]:
```
mask = do_ramp_visit['user_id'].isin(users_analyzed)

do_ramp_visit_daily = pd.pivot_table(do_ramp_visit[mask],
                                     index=␣
 ↪['user_id','role','date','membership_length'],
                                     columns = ['new_buyer'],
                                     values='buyer',
                                     aggfunc=lambda x:len(x.unique())
                                    )

do_ramp_visit_daily.rename(columns = {0:'unique_old_buyer',1:
 ↪'unique_new_buyer'},
                            inplace = True
                           )

do_ramp_visit_daily.reset_index(inplace=True)
```

```
do_ramp_visit_daily.fillna(0, inplace=True)

do_ramp_visit_daily['unique_total_buyer'] =␣
↪(do_ramp_visit_daily['unique_old_buyer'] +
                                            ␣
↪do_ramp_visit_daily['unique_new_buyer']
                                           )

do_ramp_visit_daily['%_old_buyer'] = (do_ramp_visit_daily['unique_old_buyer'] /
                                      do_ramp_visit_daily['unique_total_buyer']
                                     )

do_ramp_visit_daily['no_old_buyer_view'] =␣
↪(do_ramp_visit_daily['%_old_buyer']==0)

do_ramp_visit_daily.tail()
```

[321]:

new_buyer	user_id	role	date	membership_length	\
17208	999	Middle Man	2020-12-22	239	
17209	999	Middle Man	2021-01-12	260	
17210	999	Middle Man	2021-01-13	261	
17211	999	Middle Man	2021-01-14	262	
17212	999	Middle Man	2021-01-16	264	

new_buyer	unique_old_buyer	unique_new_buyer	unique_total_buyer	\
17208	0.0	3.0	3.0	
17209	0.0	8.0	8.0	
17210	1.0	11.0	12.0	
17211	1.0	8.0	9.0	
17212	0.0	6.0	6.0	

new_buyer	%_old_buyer	no_old_buyer_view
17208	0.000000	True
17209	0.000000	True
17210	0.083333	False
17211	0.111111	False
17212	0.000000	True

4 Exporting Table

[331]:
```
output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 4 output/
↪'
```

[332]:
```
#connectedness_village_df.to_excel(f"{output_path}" +"connectedness_village_df.
↪xlsx")
#village_neighbors_df.to_excel(f"{output_path}" +"village_neighbors_df.xlsx")
```

[333]:
```
counterfactual_price.to_excel(f"{output_path}" +"counterfactual_price_complete.
↪xlsx")
buyers.to_excel(f"{output_path}" +'buyers_complete.xlsx')
counterfactual_price_buyer_receipt.to_excel(f"{output_path}"␣
↪+'counterfactual_price_buyer_receipt_complete.xlsx')
users_do_ramp.to_excel(f"{output_path}" +"users_do_ramp_complete.xlsx")
do_ramp_visit_daily.to_excel(f"{output_path}" +"do_ramp_visit_daily_complete.
↪xlsx")
do_ramp_visit.to_excel(f"{output_path}" +"do_ramp_visit_complete.xlsx")
#transactions_sql.to_excel(f"{output_path}" +"transactions_sql_cleaned.xlsx")
last_most_transaction_do_ramp.to_excel(f"{output_path}"␣
↪+"last_most_transaction_do_ramp_complete.xlsx")
special_price_daily.to_excel(f"{output_path}" +"special_price_daily.xlsx")
price_group_user.to_excel(f"{output_path}" +"price_group_user_complete.xlsx")
```

[334]:
```
fruit_sale.to_excel(f"{output_path}" +"fruit_sale_notebook4output.xlsx")
transactions_sql.to_excel(f"{output_path}" +"transactions_sql_notebook4output.
↪xlsx")
users_receipt_duplicates.to_excel(f"{output_path}"␣
↪+"users_receipt_duplicates_notebook4output.xlsx")
```

[ ]:
```

```

Appendix B

Reproduction Code : Notebook 5 Receipt Cleaning

Notebook 5 Receipt Cleaning

August 29, 2021

1 Importing Library

[1]:
```
import pandas as pd
import numpy as np
import scipy
from numpy import nan
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from bokeh.plotting import figure
from bokeh.io import output_file, show, push_notebook, output_notebook
from bokeh.models import ColumnDataSource, Select
from ipywidgets import interact
import scipy.stats as st
import sys
import geopandas as gpd
import descartes
from shapely.geometry import Point, Polygon
import random
from scipy import stats
import statsmodels.api as sm
import folium
import json
import os
from IPython.display import HTML, display
import geopandas as gpd
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from numpy import array
import matplotlib.dates as mdates
from statsmodels.distributions.empirical_distribution import ECDF
import geopy.distance
from scipy.spatial import distance_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import networkx as nx
```

```
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
from sklearn.cluster import KMeans
from sklearn.neighbors import␣
↪(NeighborhoodComponentsAnalysis,KNeighborsClassifier)
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from community import community_louvain
from scipy import stats
import geopy.distance
from scipy.spatial import distance_matrix
import math
import shapely.wkt

pd.options.mode.chained_assignment = None
```

[2]:
```
def isnan(value):
    try:
        import math
        return math.isnan(float(value))
    except:
        return False
```

2 Importing data

[3]:
```
raw_data = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Raw Data/'

notebook_1_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 1␣
↪output/'

notebook_2_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 2␣
↪output/'

notebook_3_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 3␣
↪output/'

notebook_4_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 4␣
↪output/'
```

2.1 Importing receipt data

[4]:
```
receipt = pd.read_csv(f"{raw_data}"+"receipt.csv")

receipt.head()
```

[4]:
```
   id trans_type  mill_id  lr_id  do_id  lrm_id  user_id  \
0   9       mill      2.0    NaN    NaN     NaN       35
1  10       mill      2.0    NaN    NaN     NaN       35
2  11       mill      2.0    NaN    NaN     NaN       35
3  12       mill      2.0    NaN    NaN     NaN       35
4  13       mill      2.0    NaN    NaN     NaN       35

                                      weight_receipt  \
0  weight_receipt/dc380454cecf3fa78695cb24624d82b…
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

                                    price_receipt  status deleted_at  \
0                                             NaN       1        NaN
1  price_receipt/ef194e440719272468a5f7dd01658e5f…       1        NaN
2  price_receipt/f465dece002eb7dcccc594758ca85d61…       1        NaN
3  price_receipt/ed70501db27800629249be5afbf807f4…       1        NaN
4  price_receipt/bed1d69ab9136e8d3e39d06e805cd638…       1        NaN

              created_at             updated_at client_time_stamp  \
0  2018-12-05 15:37:29.0  2018-12-05 15:37:29.0               NaN
1  2018-12-05 15:44:07.0  2018-12-05 15:44:07.0               NaN
2  2018-12-05 15:46:43.0  2018-12-05 15:46:43.0               NaN
3  2018-12-05 19:28:59.0  2018-12-05 19:28:59.0               NaN
4  2018-12-05 19:31:50.0  2018-12-05 19:31:50.0               NaN

   update_weight_receipt  update_price_receipt       day app_version
0                      1                     0     17870         NaN
1                      0                     1     17870         NaN
2                      0                     2     17870         NaN
3                      0                     1  20181205         NaN
4                      0                     2  20181205         NaN
```

[5]:
```
receipt['do_id'].unique()
```

[5]:
```
array([nan])
```

[6]:
```
receipt['date'] = pd.to_datetime(receipt['created_at']).dt.date

mask = ~(receipt['mill_id'].isna())
receipt['mill_id'][mask]= [str(round(row)) for row in
                           receipt['mill_id'][mask]
                          ]

mask = ~(receipt['do_id'].isna())
```

```
receipt['do_id'][mask]= [str(round(row)) for row in
                         receipt['do_id'][mask]
                        ]

mask = ~(receipt['lr_id'].isna())
receipt['lr_id'][mask]= [str(round(row)) for row in
                         receipt['lr_id'][mask]
                        ]

receipt['buyer_receipt'] = (receipt['trans_type'] + '//' +
                            receipt['mill_id'].astype(str) + '//' +
                            'nan' + '//' +
                            receipt['lr_id'].astype(str)
                            )

receipt['receipt_id'] = receipt['id']
```

[7]:
```
receipt['user_id'] = receipt['user_id'].astype(str).replace('\.0', '',␣
↪regex=True)
```

[8]:
```
#receipt['buyer'] = receipt['buyer_receipt']
```

[9]:
```
mask =  receipt['buyer_receipt']=='lr//nan//nan//nan'

receipt['buyer_receipt'][mask]
```

[9]:
```
Series([], Name: buyer_receipt, dtype: object)
```

2.2 Import do_ramp_visit

[10]:
```
do_ramp_visit = pd.read_excel(f"{notebook_4_output}"+'do_ramp_visit_complete.
↪xlsx')

do_ramp_visit.drop(columns = ['Unnamed: 0'], inplace =True)

do_ramp_visit['date'] = pd.to_datetime(do_ramp_visit['date']).dt.date

do_ramp_visit.head()
```

[10]:

	id	user_id	do_id	date_time	date	month	week
0	3	124	NaN	2019-05-09 03:28:18.000000	2019-05-09	2019-05	19
1	5	201	NaN	2019-05-09 06:27:29.000000	2019-05-09	2019-05	19
2	6	201	NaN	2019-05-09 06:27:33.000000	2019-05-09	2019-05	19
3	7	201	NaN	2019-05-09 06:27:35.000001	2019-05-09	2019-05	19
4	8	201	NaN	2019-05-09 06:27:39.000000	2019-05-09	2019-05	19

```
   biweek  triweek  quadweek  …  village_cleaned      kecamatan  \
0      10        7         5  …         Seberida  Batang Gansal
1      10        7         5  …         Seberida  Batang Gansal
2      10        7         5  …         Seberida  Batang Gansal
3      10        7         5  …         Seberida  Batang Gansal
4      10        7         5  …         Seberida  Batang Gansal

            kecamatan_desa membership_length new_buyer  \
0  BATANG GANSAL__SEBERIDA               107       1.0
1  BATANG GANSAL__SEBERIDA                28       1.0
2  BATANG GANSAL__SEBERIDA                28       1.0
3  BATANG GANSAL__SEBERIDA                28       1.0
4  BATANG GANSAL__SEBERIDA                28       1.0

          buyer_kecamatan_desa kecamatan_desa_connected same_village  \
0                          NaN                      NaN          NaN
1    SEBERIDA__PANGKALAN KASAI                      0.0          0.0
2   BATANG CENAKU__BUKIT LIPAI                      0.0          0.0
3      BATANG CENAKU__AUR CINA                      0.0          0.0
4  RENGAT BARAT__PEMATANG REBA                      0.0          0.0

       user_kecamatan_desa  shortest_path_length
0                      NaN                   NaN
1  BATANG GANSAL__SEBERIDA                   3.0
2  BATANG GANSAL__SEBERIDA                   3.0
3  BATANG GANSAL__SEBERIDA                   2.0
4  BATANG GANSAL__SEBERIDA                   4.0

[5 rows x 51 columns]
```

[11]:
```
do_ramp_visit['user_id'] = do_ramp_visit['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)
```

2.3 Importing do_ramp_visit_daily data

[12]:
```
do_ramp_visit_daily = pd.
↪read_excel(f"{notebook_4_output}"+'do_ramp_visit_daily_complete.xlsx')

do_ramp_visit_daily.drop(columns = ['Unnamed: 0'], inplace =True)

do_ramp_visit_daily['date'] = pd.to_datetime(do_ramp_visit_daily['date']).dt.
↪date

do_ramp_visit_daily.head()
```

[12]:
```
   user_id        role        date  membership_length  unique_old_buyer  \
0      100  Middle Man  2019-11-05                297                 0
1      100  Middle Man  2019-11-07                299                 0
2      100  Middle Man  2019-11-08                300                 0
3      100  Middle Man  2019-11-09                301                 0
4      100  Middle Man  2019-11-10                302                 0

   unique_new_buyer  unique_total_buyer  %_old_buyer  no_old_buyer_view
0                15                  15          0.0               True
1                 6                   6          0.0               True
2                 2                   2          0.0               True
3                12                  12          0.0               True
4                 3                   3          0.0               True
```

[13]:
```
do_ramp_visit_daily['user_id'] = do_ramp_visit_daily['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

2.4 Import transactions_sql

[14]:
```
transactions_sql = pd.
↪read_excel(f"{notebook_4_output}"+'transactions_sql_notebook4output.xlsx')

transactions_sql.drop(columns = ['Unnamed: 0'], inplace =True)

transactions_sql.head()
```

[14]:
```
   index  id trans_type  mill_id  lr_id  do_id  lrm_id  user_id  grade  \
0      0  49       mill      6.0    NaN   13.0     NaN       23    NaN
1      1  50       mill      2.0    NaN   15.0     NaN       25    NaN
2      2  51       mill      6.0    NaN   13.0     NaN       23    NaN
3      3  52       mill      5.0    NaN   14.0     NaN       24    NaN
4      4  53       mill      3.0    NaN   16.0     NaN       27    NaN

   bridge  …  prev_price_diff  next_price  next_price_date  next_price_diff  \
0     NaN  …              NaN       990.0       2018-12-03             15.0
1     NaN  …              NaN       920.0       2018-12-04              0.0
2     NaN  …             15.0      1005.0       2018-12-05             15.0
3     NaN  …              NaN       980.0       2018-12-05             20.0
4     NaN  …              NaN      1065.0       2018-12-04              0.0

                  price_id  prev_price_diff_%  next_price_diff_%  outlier  \
0  transactions_price_id_49                NaN           1.538462    False
1  transactions_price_id_50                NaN           0.000000    False
2  transactions_price_id_51           1.515152           1.515152    False
3  transactions_price_id_52                NaN           2.083333    False
4  transactions_price_id_53                NaN           0.000000    False
```

```
   special_price_daily_index special_price_case
0                        NaN              False
1                        NaN              False
2                        NaN              False
3                        NaN              False
4                        NaN              False

[5 rows x 81 columns]
```

[15]:
```
transactions_sql['date'] = pd.to_datetime(transactions_sql['date']).dt.date
```

[16]:
```
mask = ~(transactions_sql['do_id'].isna())
transactions_sql['do_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['do_id'][mask]
                                  ]

mask = ~(transactions_sql['mill_id'].isna())
transactions_sql['mill_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['mill_id'][mask]
                                  ]

mask = ~(transactions_sql['lr_id'].isna())
transactions_sql['lr_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['lr_id'][mask]
                                  ]

mask = ~(transactions_sql['lrm_id'].isna())
transactions_sql['lrm_id'][mask] = [str(round(row)) for row in
                                   transactions_sql['lrm_id'][mask]
                                  ]

transactions_sql['user_id'] = transactions_sql['user_id'].astype(str).
 ↪replace('\.0', '', regex=True)
```

[17]:
```
mask =  transactions_sql['buyer_receipt']=='lr//nan//nan//nan'

transactions_sql['buyer_receipt'][mask]
```

[17]:
```
Series([], Name: buyer_receipt, dtype: object)
```

[18]:
```
mask =  transactions_sql['buyer']=='lr//nan//nan//nan'

transactions_sql['buyer'][mask]
```

[18]:
```
Series([], Name: buyer, dtype: object)
```

2.5 Importing users data

[19]:
```
users = pd.read_csv(f"{notebook_1_output}" + "users_processed.csv")
```

[20]:
```
users.replace({'NAN':np.nan}, inplace=True)
```

[21]:
```
users['kecamatan_desa'] = users['kecamatan'] + '__' + users['village_cleaned']

users['kecamatan_desa'] = users['kecamatan_desa'].str.upper()

users['user_kecamatan_desa']  = users['kecamatan_desa']
```

[22]:
```
users['date'] = pd.to_datetime(users['date']).dt.date
```

[23]:
```
users.columns
```

[23]:
```
Index(['id', 'name', 'nickname', 'village', 'village_id', 'photo',
       'analysis_status', 'analysis_note', 'phone', 'country_code',
       'phone_verified_at', 'role_id', 'sms_notification', 'push_notification',
       'status', 'src', 'referral', 'password', 'addby', 'remember_token',
       'deleted_at', 'created_at', 'updated_at', 'client_time_stamp',
       'user_id', 'village_cleaned', 'sub_district_id', 'kecamatan', 'role',
       'date', 'month', 'week', 'biweek', 'triweek', 'quadweek', 'year',
       'year_week', 'year_biweek', 'year_triweek', 'year_quadweek',
       'kecamatan_desa', 'user_kecamatan_desa'],
      dtype='object')
```

[24]:
```
users['user_id'] = users['user_id'].astype(str).replace('\.0', '', regex=True)
```

2.6 Import counterfactual_price data

[25]:
```
counterfactual_price = pd.read_excel(f"{notebook_4_output}"␣
↪+"counterfactual_price_complete.xlsx")

counterfactual_price.drop(columns = ['Unnamed: 0'], inplace =True)

counterfactual_price['date'] = pd.to_datetime(counterfactual_price['date']).dt.
↪date

counterfactual_price.head()
```

[25]:

	date	buyer	buyer_receipt	grade_cleaned	price_cleaned	\
0	2018-12-01	lr//nan//nan//11	NaN	NaN	NaN	
1	2018-12-02	lr//nan//nan//11	NaN	NaN	NaN	
2	2018-12-03	lr//nan//nan//11	NaN	NaN	NaN	
3	2018-12-04	lr//nan//nan//11	NaN	NaN	NaN	
4	2018-12-05	lr//nan//nan//11	NaN	NaN	NaN	

```
   max_price_cleaned  min_grade_cleaned  grade_cleaned_buyer_receipt  \
0                NaN                NaN                          NaN
1                NaN                NaN                          NaN
2                NaN                NaN                          NaN
3                NaN                NaN                          NaN
4                NaN                NaN                          NaN

   price_cleaned_buyer_receipt  max_price_cleaned_buyer_receipt  \
0                          NaN                              NaN
1                          NaN                              NaN
2                          NaN                              NaN
3                          NaN                              NaN
4                          NaN                              NaN

   min_grade_cleaned_buyer_receipt  price_counterfactual_table  \
0                              NaN                         NaN
1                              NaN                         NaN
2                              NaN                         NaN
3                              NaN                         NaN
4                              NaN                         NaN

   grade_counterfactual_table  post_grading_price_counterfactual_table  \
0                         NaN                                      NaN
1                         NaN                                      NaN
2                         NaN                                      NaN
3                         NaN                                      NaN
4                         NaN                                      NaN

   max_price_counterfactual_table  min_grade_counterfactual_table  \
0                             NaN                             NaN
1                             NaN                             NaN
2                             NaN                             NaN
3                             NaN                             NaN
4                             NaN                             NaN

   max_post_grading_price_counterfactual_table buyer_kecamatan_desa
0                                          NaN                  NaN
1                                          NaN                  NaN
2                                          NaN                  NaN
3                                          NaN                  NaN
4                                          NaN                  NaN
```

2.7 Import counterfactual_price_buyer_receipt data

[26]:
```
counterfactual_price_buyer_receipt = pd.read_excel(f"{notebook_4_output}"␣
↪+'counterfactual_price_buyer_receipt_complete.xlsx')

counterfactual_price_buyer_receipt.drop(columns = ['Unnamed: 0'], inplace =True)

counterfactual_price_buyer_receipt['date'] = pd.
↪to_datetime(counterfactual_price_buyer_receipt['date']).dt.date

counterfactual_price_buyer_receipt.head()
```

[26]:

	date	buyer_receipt	grade_cleaned_buyer_receipt	price_cleaned_buyer_receipt	max_price_cleaned_buyer_receipt	min_grade_cleaned_buyer_receipt
0	2018-12-01	lr//nan//nan//11	NaN	NaN	NaN	NaN
1	2018-12-02	lr//nan//nan//11	NaN	NaN	NaN	NaN
2	2018-12-03	lr//nan//nan//11	NaN	NaN	NaN	NaN
3	2018-12-04	lr//nan//nan//11	NaN	NaN	NaN	NaN
4	2018-12-05	lr//nan//nan//11	NaN	NaN	NaN	NaN

2.8 Import users_do_ramp data

[27]:
```
users_do_ramp = pd.read_excel(f"{notebook_4_output}" +'users_do_ramp_complete.
↪xlsx')

users_do_ramp.drop(columns = ['Unnamed: 0'], inplace =True)

users_do_ramp.head()
```

[27]:

	user_id	do_id	lrm_id	mill_id	lr_id	buyer	trans_type
0	553.0	21.0	NaN	1.0	NaN	mill//1//21//nan	mill
1	554.0	45.0	NaN	17.0	NaN	mill//17//45//nan	mill
2	554.0	185.0	NaN	33.0	NaN	mill//33//185//nan	mill
3	555.0	21.0	NaN	1.0	NaN	mill//1//21//nan	mill

```
4    556.0    NaN    40.0      NaN   45.0    lr//nan//nan//45         lr

          source       buyer_receipt
0  in-app-survey   mill//1//nan//nan
1  in-app-survey  mill//17//nan//nan
2  in-app-survey  mill//33//nan//nan
3  in-app-survey   mill//1//nan//nan
4  in-app-survey    lr//nan//nan//45
```

[28]:
```
mask = users_do_ramp['buyer_receipt'] == 'lr//nan//nan//nan'

users_do_ramp[mask]
```

[28]:
```
Empty DataFrame
Columns: [user_id, do_id, lrm_id, mill_id, lr_id, buyer, trans_type, source,
buyer_receipt]
Index: []
```

[29]:
```
users_do_ramp['mill_id'] = users_do_ramp['mill_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_do_ramp['lr_id'] = users_do_ramp['lr_id'].astype(str).replace('\.0', '',␣
↪regex=True)

users_do_ramp['do_id'] = users_do_ramp['do_id'].astype(str).replace('\.0', '',␣
↪regex=True)

users_do_ramp['lrm_id'] = users_do_ramp['lrm_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_do_ramp['user_id'] = users_do_ramp['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_do_ramp.replace('nan',np.nan, inplace=True)
```

2.9 Import buyers data

[30]:
```
buyers = pd.read_excel(f"{notebook_4_output}" +'buyers_complete.xlsx')

buyers.drop(columns = ['Unnamed: 0'], inplace =True)

#buyers['date'] = pd.to_datetime(buyers['date']).dt.date

buyers.head()
```

[30]:
```
   Unnamed: 0.1  user_id            buyer    buyer_receipt         lng  \
0             0      202  lr//nan//nan//1  lr//nan//nan//1  102.668298
1             1      203  lr//nan//nan//2  lr//nan//nan//2  102.514567
2             2      204  lr//nan//nan//3  lr//nan//nan//3  102.423992
3             3      205  lr//nan//nan//4  lr//nan//nan//4  102.412089
4             4      218  lr//nan//nan//5  lr//nan//nan//5  102.257117

        lat        buyer_kecamatan_desa counterfactual_buyer  \
0 -0.889027          KEMUNING__KERITANG      lr//nan//nan//1
1 -0.734371     BATANG GANSAL__SEBERIDA      lr//nan//nan//2
2 -0.628439   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//3
3 -0.551158   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//4
4 -0.658824  BATANG CENAKU__BUKIT LIPAI      lr//nan//nan//5

  counterfactual_buyer_kecamatan_desa
0                  KEMUNING__KERITANG
1              BATANG GANSAL__SEBERIDA
2            SEBERIDA__PANGKALAN KASAI
3            SEBERIDA__PANGKALAN KASAI
4           BATANG CENAKU__BUKIT LIPAI
```

[31]:
```
buyers['user_id'] = buyers['user_id'].astype(str).replace('\.0', '', regex=True)
```

2.10 Import last__most__transaction__do__ramp

[32]:
```
last_most_transaction_do_ramp = pd.read_excel(f"{notebook_4_output}"␣
↪+'last_most_transaction_do_ramp_complete.xlsx')

last_most_transaction_do_ramp.drop(columns = ['Unnamed: 0'], inplace =True)

last_most_transaction_do_ramp.head()
```

[32]:
```
    id     do_name ramp_manager_name mill_name ramp_name  user_id  do_id  \
0  1.0  TSR Guntur               NaN      SKIP       NaN    553.0   21.0
1  2.0  TSR Guntur               NaN      SKIP       NaN    553.0   21.0
2  3.0  TSR Guntur               NaN      SKIP       NaN    553.0   21.0
3  4.0          NY               NaN    Inecda       NaN    554.0   45.0
4  5.0          NY               NaN    Inecda       NaN    554.0   45.0

   ramp_manager_id  mill_id  ramp_id  …             updated_at  \
0              NaN      1.0      NaN  …  2019-11-25 14:00:32.0
1              NaN      1.0      NaN  …  2020-04-09 23:21:51.0
2              NaN      1.0      NaN  …  2020-04-09 23:21:45.0
3              NaN     17.0      NaN  …  2019-11-27 10:56:56.0
4              NaN     17.0      NaN  …  2020-04-09 23:21:58.0

       client_time_stamp             deleted_at trans_type              buyer  \
```

```
0  2019-11-25 14:00:31.0                    NaN       mill   mill//1//21//nan
1  2019-11-25 14:00:31.0  2020-04-09 23:21:51.0       mill   mill//1//21//nan
2  2019-11-25 14:00:31.0  2020-04-09 23:21:45.0       mill   mill//1//21//nan
3  2019-11-27 10:56:54.0                    NaN       mill  mill//17//45//nan
4  2019-11-27 10:56:54.0  2020-04-09 23:21:58.0       mill  mill//17//45//nan

        buyer_receipt unknown_do  buyer_in_pempem           role  \
0   mill//1//nan//nan      False             True  Amprah Farmer
1   mill//1//nan//nan      False             True  Amprah Farmer
2   mill//1//nan//nan      False             True  Amprah Farmer
3  mill//17//nan//nan      False             True     Middle Man
4  mill//17//nan//nan      False             True     Middle Man

          source
0  in-app-survey
1  in-app-survey
2  in-app-survey
3  in-app-survey
4  in-app-survey

[5 rows x 21 columns]
```

[33]:
```
last_most_transaction_do_ramp.shape
```

[33]:
```
(1674, 21)
```

[34]:
```
last_most_transaction_do_ramp['user_id'] =␣
↪last_most_transaction_do_ramp['user_id'].astype(str).replace('\.0', '',␣
↪regex=True)

mask = ~(last_most_transaction_do_ramp['do_id'].isna())
last_most_transaction_do_ramp['do_id'][mask] = [str(round(row)) for row in
                                                ␣
↪last_most_transaction_do_ramp['do_id'][mask]
                                                ]

mask = ~(last_most_transaction_do_ramp['ramp_manager_id'].isna())
last_most_transaction_do_ramp['ramp_manager_id'][mask] = [str(round(row)) for␣
↪row in
                                                          ␣
↪last_most_transaction_do_ramp['ramp_manager_id'][mask]
                                                          ]

mask = ~(last_most_transaction_do_ramp['ramp_id'].isna())
last_most_transaction_do_ramp['ramp_id'][mask] = [str(round(row)) for row in
                                                  ␣
↪last_most_transaction_do_ramp['ramp_id'][mask]
```

```
                                                    ]

mask = ~(last_most_transaction_do_ramp['mill_id'].isna())
last_most_transaction_do_ramp['mill_id'][mask] = [str(round(row)) for row in
                                                    ␣
 ↪last_most_transaction_do_ramp['mill_id'][mask]
                                                    ]
```

[35]:
```
users_analyzed = list(last_most_transaction_do_ramp['user_id'].unique())

len(users_analyzed)
```

[35]:
```
967
```

2.11 Importing users_location and distance data

[36]:
```
users_location = pd.read_excel(f"{notebook_2_output}" +'users_location.xlsx')

users_location.drop(columns = ['Unnamed: 0'],inplace = True)

users_location.head()
```

[36]:
```
   user_id                                all_hours_footprint  \
0       23  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1       24  LINESTRING (102.594475 -0.7465233, 102.6545117…
2       25  POLYGON ((102.5231233381282 -0.735018107855270…
3       27  POLYGON ((102.3894691781223 -0.640409193665700…
4       31  POLYGON ((102.511719261713 -0.7214787989871149…

                               night_time_footprint  \
0  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1                                                NaN
2  POLYGON ((102.5231568 -0.7350115, 102.5230697 …
3  POLYGON ((102.3458618 -0.6623945, 102.3449446 …
4  POLYGON ((102.4559663599486 -0.661479634417434…

                                          geometry  \
0  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1                                                NaN
2  POLYGON ((102.5231568 -0.7350115, 102.5230697 …
3  POLYGON ((102.3458618 -0.6623945, 102.3449446 …
4  POLYGON ((102.4559663599486 -0.661479634417434…

  night_time_footprint_type  night_time_footprint_equiv_radius  \
0                   Polygon                          10.345682
1                       NaN                                NaN
2                   Polygon                           0.105661
```

```
3                   Polygon                            5.502121
4                   Polygon                            3.801410

   night_time_footprint_area all_hours_footprint_type  \
0                 336.084051                   Polygon
1                        NaN               LineString
2                   0.035056                   Polygon
3                  95.058274                   Polygon
4                  45.375262                   Polygon

   all_hours_footprint_area  all_hours_footprint_equiv_radius  \
0                436.877202                         11.795460
1                       NaN                               NaN
2                  0.311361                          0.314896
3                112.991683                          5.998718
4                366.018211                         10.796588

  night_time_modus_coord  all_hours_modus_coord           modus_coords  \
0  (102.50811, -0.72163)  (102.50811, -0.72163)  (102.50811, -0.72163)
1                    NaN  (102.59448, -0.74652)  (102.59448, -0.74652)
2  (102.52397, -0.73443)    (102.52311, -0.735)  (102.52397, -0.73443)
3  (102.40708, -0.56157)  (102.40708, -0.56157)  (102.40708, -0.56157)
4   (102.41997, -0.6234)   (102.41997, -0.6234)   (102.41997, -0.6234)

   modus_long  modus_lat
0   102.50811   -0.72163
1   102.59448   -0.74652
2   102.52397   -0.73443
3   102.40708   -0.56157
4   102.41997   -0.62340
```

[37]:
```
users_location['user_id'] = users_location['user_id'].astype(str).replace('\.
↪0', '', regex=True)
```

[38]:
```
user_buyer_distance = pd.read_excel(f"{notebook_2_output}"␣
↪+'user_buyer_distance.xlsx')

user_buyer_distance.set_index('user_id',inplace = True)

user_buyer_distance.head()
```

[38]:
```
         lr//nan//nan//1  lr//nan//nan//102  lr//nan//nan//104  \
user_id
100            32.819371       13248.011418          24.942663
1000           53.379901       13212.817446          20.103252
1002           47.400023       13238.403718          10.859041
1004           53.553551       13212.771414          19.991087
```

```
1006           51.822446       13235.860914           8.260469

         lr//nan//nan//105  lr//nan//nan//106  lr//nan//nan//107  \
user_id
100              53.316243          26.679406           8.381088
1000             59.517287          56.221612          32.525556
1002             40.735977          28.414093           7.844243
1004             59.348067          56.192050          32.526995
1006             37.315562          30.183422          12.231705

         lr//nan//nan//109  lr//nan//nan//112  lr//nan//nan//113  \
user_id
100              34.317010          65.829474          41.436333
1000              6.355908          56.044285          45.371326
1002             29.976136          50.337231          27.471050
1004              6.581063          55.826154          45.207740
1006             30.459011          45.823785          23.644009

         lr//nan//nan//114  …  mill//76//199//nan  mill//8//191//nan  \
user_id                     …
100              40.783344  …           40.632112          38.259242
1000             47.576110  …           48.836304          32.390252
1002             27.681978  …           28.041537          44.390068
1004             47.422697  …           48.687886          32.613463
1006             24.243105  …           24.822301          47.236948

         mill//8//193//nan  mill//8//197//nan  mill//8//198//nan  \
user_id
100              49.010265          34.348542          48.471624
1000             36.980635          30.610300          36.982399
1002             53.796389          40.759999          53.431403
1004             37.202969          30.828904          37.205407
1006             56.189899          43.733861          55.871841

         mill//8//27//nan  mill//8//38//nan  mill//8//39//nan  \
user_id
100             36.467754         23.104722         37.851248
1000            31.374704         32.832156         32.195327
1002            42.658988         32.978098         44.012479
1004            31.596305         33.012594         32.418198
1006            45.549363         36.771086         46.872537

         mill//92//203//nan  mill//98//205//nan
user_id
100               56.388654           53.055187
1000              72.481894           67.928947
1002              48.262965           44.172875
```

```
1004              72.350161           67.793946
1006              46.353225           42.066089

[5 rows x 126 columns]
```

[39]:
```
user_buyer_receipt_distance = pd.read_excel(f"{notebook_2_output}"␣
↪+'user_buyer_receipt_distance.xlsx')

user_buyer_receipt_distance.set_index('user_id',inplace = True)

user_buyer_receipt_distance.head()
```

[39]:

user_id	lr//nan//nan//1	lr//nan//nan//104	lr//nan//nan//105	\
100	32.819371	24.942663	53.316243	
1000	53.379901	20.103252	59.517287	
1002	47.400023	10.859041	40.735977	
1004	53.553551	19.991087	59.348067	
1006	51.822446	8.260469	37.315562	

user_id	lr//nan//nan//106	lr//nan//nan//107	lr//nan//nan//109	\
100	26.679406	8.381088	34.317010	
1000	56.221612	32.525556	6.355908	
1002	28.414093	7.844243	29.976136	
1004	56.192050	32.526995	6.581063	
1006	30.183422	12.231705	30.459011	

user_id	lr//nan//nan//112	lr//nan//nan//113	lr//nan//nan//114	\
100	65.829474	41.436333	40.783344	
1000	56.044285	45.371326	47.576110	
1002	50.337231	27.471050	27.681978	
1004	55.826154	45.207740	47.422697	
1006	45.823785	23.644009	24.243105	

user_id	lr//nan//nan//115	…	mill//4//nan//nan	mill//42//nan//nan	\
100	33.673207	…	19.604971	35.695854	
1000	3.633534	…	34.980984	49.693267	
1002	25.142242	…	31.153034	48.690800	
1004	3.467859	…	35.143501	49.887495	
1006	24.355984	…	35.237575	52.855623	

user_id	mill//47//nan//nan	mill//5//nan//nan	mill//6//nan//nan	\
100	39.359800	21.922404	10.465061	

```
1000              14.454261          33.749129          34.716556
1002              37.559532          32.446721          23.726118
1004              14.666958          33.923503          34.832500
1006              38.536250          36.355382          28.120624

         mill//7//nan//nan  mill//76//nan//nan  mill//8//nan//nan  \
user_id
100              21.810801          40.632112          37.851248
1000             19.688154          48.836304          32.195327
1002              8.932565          28.041537          44.012479
1004             19.611590          48.687886          32.418198
1006              7.854885          24.822301          46.872537

         mill//92//nan//nan  mill//98//nan//nan
user_id
100               56.388654           53.055187
1000              72.481894           67.928947
1002              48.262965           44.172875
1004              72.350161           67.793946
1006              46.353225           42.066089

[5 rows x 99 columns]
```

[40]:
```
user_buyer_distance.reset_index(inplace=True)
```

[41]:
```
user_buyer_distance['user_id'] = user_buyer_distance['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[42]:
```
user_buyer_distance.set_index('user_id', inplace = True)

user_buyer_distance
```

[42]:
```
         lr//nan//nan//1  lr//nan//nan//102  lr//nan//nan//104  \
user_id
100            32.819371       13248.011418          24.942663
1000           53.379901       13212.817446          20.103252
1002           47.400023       13238.403718          10.859041
1004           53.553551       13212.771414          19.991087
1006           51.822446       13235.860914           8.260469
…                    …                  …                  …
994            51.545121       13212.530642          22.729959
995            51.108713       13234.252843           6.495181
996            60.867849       13225.898407           7.223441
997            56.225403       13224.694771           3.199648
999            49.431588       13238.189530          10.429468

         lr//nan//nan//105  lr//nan//nan//106  lr//nan//nan//107  \
```

```
user_id
100              53.316243          26.679406           8.381088
1000             59.517287          56.221612          32.525556
1002             40.735977          28.414093           7.844243
1004             59.348067          56.192050          32.526995
1006             37.315562          30.183422          12.231705
…                    …                  …                  …
994              62.659944          57.718306          33.658868
995              39.015007          31.978866          12.582664
996              35.322374          39.948713          23.327048
997              41.303786          41.356501          21.442235
999              38.684174          28.148744           9.391694

         lr//nan//nan//109  lr//nan//nan//112  lr//nan//nan//113  \
user_id
100              34.317010          65.829474          41.436333
1000              6.355908          56.044285          45.371326
1002             29.976136          50.337231          27.471050
1004              6.581063          55.826154          45.207740
1006             30.459011          45.823785          23.644009
…                    …                  …                  …
994               3.200070          59.481023          48.467602
995              28.308669          46.424704          25.166843
996              29.163861          37.121125          21.120619
997              23.171375          42.680595          27.044693
999              31.072259          48.397376          25.380492

         lr//nan//nan//114  …  mill//76//199//nan  mill//8//191//nan  \
user_id                     …
100              40.783344  …           40.632112          38.259242
1000             47.576110  …           48.836304          32.390252
1002             27.681978  …           28.041537          44.390068
1004             47.422697  …           48.687886          32.613463
1006             24.243105  …           24.822301          47.236948
…                    …  …                  …                  …
994              50.573011  …           51.782217          29.285626
995              25.971718  …           26.637920          45.452621
996              23.360809  …           24.703331          51.033505
997              29.027852  …           30.229992          44.931851
999              25.621516  …           26.011356          46.288945

         mill//8//193//nan  mill//8//197//nan  mill//8//198//nan  \
user_id
100              49.010265          34.348542          48.471624
1000             36.980635          30.610300          36.982399
1002             53.796389          40.759999          53.431403
1004             37.202969          30.828904          37.205407
```

```
1006             56.189899          43.733861          55.871841
…                      …                  …                  …
994              33.573458          27.701392          33.585966
995              54.264683          41.999288          53.958714
996              58.477454          48.002520          58.296182
997              52.295588          41.963519          52.115009
999              55.578639          42.687838          55.226765

         mill//8//27//nan  mill//8//38//nan  mill//8//39//nan  \
user_id
100              36.467754         23.104722         37.851248
1000             31.374704         32.832156         32.195327
1002             42.658988         32.978098         44.012479
1004             31.596305         33.012594         32.418198
1006             45.549363         36.771086         46.872537
…                      …                 …                 …
994              28.347605         30.906569         29.108429
995              43.782249         35.452104         45.093154
996              49.522507         43.666185         50.717546
997              43.443090         38.241684         44.621431
999              44.567676         35.053256         45.914433

         mill//92//203//nan  mill//98//205//nan
user_id
100                56.388654           53.055187
1000               72.481894           67.928947
1002               48.262965           44.172875
1004               72.350161           67.793946
1006               46.353225           42.066089
…                        …                   …
994                75.237815           70.722729
995                48.439231           44.127002
996                48.733304           44.090767
997                53.868948           49.298922
999                46.555507           42.405887

[1246 rows x 126 columns]
```

[43]:
```
user_to_user_distance = pd.read_excel(f"{notebook_2_output}"␣
↪+'user_to_user_distance.xlsx')

user_to_user_distance.set_index('user_id_x',inplace = True)

user_to_user_distance.head()
```

[43]:
```
                100       1000       1002       1004       1006       101  \
user_id_x
```

```
100          0.000000  35.212250  15.688990  35.266024  20.227828   8.943032
1000        35.212250   0.000000  27.965310   0.225160  27.529178  34.601300
1002        15.688990  27.965310   0.000000  27.919442   4.546576  22.437666
1004        35.266024   0.225160  27.919442   0.000000  27.448058  34.708699
1006        20.227828  27.529178   4.546576  27.448058   0.000000  26.868536

                1010       1011       1015       1016  …        989  \
user_id_x                                              …
100         15.522704  28.206003  17.726662  20.296788  …  38.743395
1000        25.291195  29.010123  27.753069  27.711696  …  36.918778
1002         2.922928  12.554371   2.047333   4.627839  …  23.394539
1004        25.255252  28.874066  27.691213  27.629963  …  36.743359
1006         5.909832   8.012738   2.501799   0.199257  …  18.963607

                  99        990        991        993        994        995  \
user_id_x
100          3.750649  35.483473   7.075746  52.953666  35.554443  20.165157
1000        33.647678  12.047019  36.567522  39.136045   3.439225  25.401726
1002        12.047699  33.542645  21.908107  57.301075  29.762790   4.764004
1004        33.679609  12.271303  36.662343  39.354252   3.655598  25.323307
1006        16.565677  34.616438  26.429360  59.543478  29.766126   2.152925

                996        997        999
user_id_x
100         30.866922  27.986876  17.499441
1000        24.261539  18.640963  28.679458
1002        15.483276  14.048443   2.091795
1004        24.101387  18.501366  28.618370
1006        11.312642  11.188134   2.876087

[5 rows x 1246 columns]
```

2.12 Importing app_user_extra_infos

[44]:
```
app_user_extra_infos = (pd.read_csv(f"{notebook_1_output}"␣
↪+'app_user_extra_infos.csv', delimiter = ','))
```

[45]:
```
app_user_extra_infos.head()
```

[45]:
```
   id  user_id  last_transaction_time       last_logged_time  \
0   1        1  2019-01-08 06:06:03.0  2019-05-07 00:00:00.0
1   2       23  2020-05-12 21:28:45.0  2020-05-14 21:27:03.0
2   3       24  2019-06-11 16:48:51.0  2019-05-26 08:59:19.0
3   4       25  2020-03-11 22:27:26.0  2020-04-01 21:32:29.0
4   5       27  2020-05-17 02:06:16.0  2020-05-17 02:06:06.0

      last_activity_time  last_notification_time app_version  is_app_admin  \
```

```
0  2019-05-08 17:59:07.0                    NaN         NaN             0
1  2020-05-14 21:27:05.0                    NaN       2.6.1             0
2  2019-05-26 09:00:03.0                    NaN        2.10             0
3  2020-04-04 21:18:59.0                    NaN       2.4.0             0
4  2020-05-22 00:31:13.0                    NaN       2.6.0             0

   week2sms  week4sms          week2sms_time  week4sms_time  \
0         0         0                    NaN            NaN
1         0         0                    NaN            NaN
2         1         0  2019-06-25 09:00:02.0            NaN
3         0         0                    NaN            NaN
4         0         0                    NaN            NaN

   privacypolicy_version  incentivepolicy_version  termsconditions_version  \
0                      0                        0                        0
1                      4                        2                        3
2                      3                        2                        3
3                      4                        2                        3
4                      4                        2                        3

              created_at             updated_at deleted_at
0  2018-11-30 22:46:47.0                    NaN        NaN
1  2018-12-01 19:14:06.0  2020-05-14 21:27:05.0        NaN
2  2018-12-02 15:20:16.0  2019-06-25 09:00:02.0        NaN
3  2018-12-02 22:41:54.0  2020-04-04 21:18:59.0        NaN
4  2018-12-03 19:50:15.0  2020-05-22 00:31:13.0        NaN
```

[46]:
```
app_user_extra_infos['user_id'] = app_user_extra_infos['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

2.13 Importing price_group_user

[47]:
```
price_group_user = pd.
↪read_excel(f"{notebook_4_output}"+'price_group_user_complete.xlsx')

price_group_user['user_id'] = price_group_user['user_id'].astype(str).
↪replace('\.0', '', regex=True)

price_group_user.tail()
```

[47]:
```
     Unnamed: 0  price_group_user_id  price_group_id user_id  \
758         758                  759             101    2194
759         759                  760             101    2195
760         760                  761             101    2196
761         761                  762             101    1702
762         762                  763             101    2197
```

```
         client_time_stamp             created_at             updated_at  \
758  2021-01-27 21:57:53.0  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0
759  2021-01-27 21:57:53.0  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0
760  2021-01-27 21:57:53.0  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0
761  2021-01-27 21:57:53.0  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0
762  2021-01-27 21:57:53.0  2021-01-27 21:57:54.0  2021-01-27 21:57:54.0

    deleted_at       date date_added           role  added_by_user_id  \
758        NaN 2021-01-27 2021-01-27  Amprah Farmer              1701
759        NaN 2021-01-27 2021-01-27     Middle Man              1701
760        NaN 2021-01-27 2021-01-27     Middle Man              1701
761        NaN 2021-01-27 2021-01-27  Amprah Farmer              1701
762        NaN 2021-01-27 2021-01-27  Amprah Farmer              1701

                 buyer      buyer_receipt
758  lr//nan//nan//121  lr//nan//nan//121
759  lr//nan//nan//121  lr//nan//nan//121
760  lr//nan//nan//121  lr//nan//nan//121
761  lr//nan//nan//121  lr//nan//nan//121
762  lr//nan//nan//121  lr//nan//nan//121
```

[48]:
```
cols = ['price_group_user_id','price_group_id','user_id']

price_group_user[cols].drop_duplicates().shape
```

[48]:
```
(763, 3)
```

2.14 Importing special_price_daily table

[49]:
```
special_price_daily = pd.read_excel(f"{notebook_4_output}"+"special_price_daily.
↪xlsx")

special_price_daily['date'] = pd.to_datetime(special_price_daily['date']).dt.
↪date

special_price_daily['user_id'] = special_price_daily['user_id'].astype(str).
↪replace('\.0', '', regex=True)

special_price_daily['added_by_user_id'] =␣
↪special_price_daily['added_by_user_id'].astype(str).replace('\.0', '',␣
↪regex=True)
```

[50]:
```
special_price_daily.head()
```

[50]:
```
   Unnamed: 0  price_group_price_id  price_group_id added_by_user_id  \
0           0                   2.0               2              608
1           1                   2.0               2              608
2           2                   2.0               2              608
3           3                   2.0               2              608
4           4                   7.0               2              608

   added_by_backend_user_id         day   price  published  \
0                       NaN  20200312.0  1000.0        1.0
1                       NaN  20200312.0  1000.0        1.0
2                       NaN  20200312.0  1000.0        1.0
3                       NaN  20200312.0  1000.0        1.0
4                       NaN  20200313.0  1000.0        1.0

       client_time_stamp             created_at  …         lng       lat  \
0  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  …  102.890497 -0.602906
1  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  …  102.890497 -0.602906
2  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  …  102.890497 -0.602906
3  2020-03-11 19:08:37.0  2020-03-11 19:08:46.0  …  102.890497 -0.602906
4  2020-03-12 21:01:29.0  2020-03-12 21:01:44.0  …  102.890497 -0.602906

   buyer_kecamatan_desa  counterfactual_buyer  \
0  KEMPAS__HARAPAN TANI     mill//8//193//nan
1  KEMPAS__HARAPAN TANI     mill//8//193//nan
2  KEMPAS__HARAPAN TANI     mill//8//193//nan
3  KEMPAS__HARAPAN TANI     mill//8//193//nan
4  KEMPAS__HARAPAN TANI     mill//8//193//nan

  counterfactual_buyer_kecamatan_desa user_id  date_added  posted_before_join  \
0                KEMPAS__HARAPAN TANI     609  2020-03-11               False
1                KEMPAS__HARAPAN TANI     310  2020-03-11               False
2                KEMPAS__HARAPAN TANI     200  2020-03-11               False
3                KEMPAS__HARAPAN TANI      97  2020-03-11               False
4                KEMPAS__HARAPAN TANI     609  2020-03-11               False

                    source special_price_daily_index
0  special price membership                         0
1  special price membership                         1
2  special price membership                         2
3  special price membership                         3
4  special price membership                         4

[5 rows x 26 columns]
```

[51]:
```
special_price_daily['source'] ='special price membership'
```

[52]:
```
price_group_user = pd.
↪read_excel(f"{notebook_4_output}"+"price_group_user_complete.xlsx")

price_group_user['date'] = pd.to_datetime(price_group_user['date']).dt.date

price_group_user['user_id'] = price_group_user['user_id'].astype(str).
↪replace('\.0', '', regex=True)

price_group_user['added_by_user_id'] = price_group_user['added_by_user_id'].
↪astype(str).replace('\.0', '', regex=True)
```

2.15 Importing users_receipt_duplicates and price_imputed_manual

[53]:
```
users_receipt_duplicates = pd.
↪read_excel(f"{notebook_4_output}"+"users_receipt_duplicates_notebook4output.
↪xlsx",
                                        #sep=";"
                                       )


users_receipt_duplicates['date'] = pd.
↪to_datetime(users_receipt_duplicates['date']).dt.date

users_receipt_duplicates['imputed_post_grading_weight'].replace({0:np.
↪nan},inplace =True)

users_receipt_duplicates['imputed_price_manual'].replace({0:np.nan},inplace␣
↪=True)

users_receipt_duplicates.head()
```

[53]:

	Unnamed: 0	level_0	index	receipt_id	duplicates_final	\
0	0	0	0	30689	NaN	
1	1	1	1	30687	NaN	
2	2	2	2	30685	NaN	
3	3	3	3	30683	NaN	
4	4	4	4	30678	NaN	

	imputed_post_grading_weight	imputed_price_manual	imputed_grade_manual	\
0	3069.0	1925.0	NaN	
1	5376.0	1925.0	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	

	buyer_receipt	date	Unnamed: 7	price_id	\

```
0  mill//4//nan//nan  2020-12-01         NaN  manual_price_id_30689
1  mill//4//nan//nan  2020-12-01         NaN  manual_price_id_30687
2  mill//4//nan//nan  2020-11-30         NaN  manual_price_id_30685
3  mill//4//nan//nan  2020-11-28         NaN  manual_price_id_30683
4  lr//nan//nan//113  2020-11-27         NaN  manual_price_id_30678

   special_price_daily_index
0                    18033.0
1                    18033.0
2                    18002.0
3                    17940.0
4                        NaN
```

[54]:
```
users_receipt_duplicates['imputed_post_grading_weight'].max()
```

[54]:
```
17276.0
```

2.16 Importing fruit_sale table

[55]:
```
fruit_sale = pd.read_excel(f"{notebook_4_output}"+ "fruit_sale_notebook4output.
↪xlsx")

fruit_sale['fruit_sale_id'] = fruit_sale['fruit_sale_id'].astype(str).
↪replace('\.0', '', regex=True)

fruit_sale.head()
```

[55]:
```
   Unnamed: 0  index fruit_sale_id                fruit_sale_identifier  \
0           0      0           598  33988ff6-bf17-4a3a-a9d3-38328100142f
1           1      1           599  779c2c75-6269-4c54-8a3d-bcb0fcb430b5
2           2      2           601  c61bc660-1d5e-4992-91e2-8e4d4e8ddcc3
3           3      3           602  224a39c6-2bee-4c5a-8105-64db0314d6e7
4           4      4           604  0c137e5a-a956-43a5-a74b-0a7b55dd2846

   telemetry_log_id buyer_type  user_id  total_weight  total_sale  \
0               NaN        NaN     1585        1500.0   1500000.0
1               NaN        NaN     1583        1160.0   1650000.0
2               NaN         do     1059        1800.0         NaN
3               NaN         do       97        2000.0         NaN
4               NaN       mill     1062        7000.0         NaN

            payment_date  …  mill_id  ramp_id  do_id  ramp_manager_id  \
0  2020-10-30 00:00:00.0  …      NaN      NaN    NaN              NaN
1  2020-10-30 00:00:00.0  …      NaN      NaN    NaN              NaN
2                    NaN  …     92.0      NaN  203.0              NaN
3                    NaN  …      2.0      NaN   15.0              NaN
```

```
4                    NaN  …      1.0      NaN    NaN               NaN

   grade_fruit_sale lr_id lrm_id       buyer_receipt                buyer  \
0               NaN   NaN    NaN                 NaN                  NaN
1               NaN   NaN    NaN                 NaN                  NaN
2              4.44   NaN    NaN  mill//92//nan//nan  mill//92//203//nan
3              6.00   NaN    NaN   mill//2//nan//nan    mill//2//15//nan
4              5.00   NaN    NaN   mill//1//nan//nan   mill//1//nan//nan

  special_price_daily_index
0                       NaN
1                       NaN
2                       NaN
3                       NaN
4                       NaN

[5 rows x 35 columns]
```

[56]:
```
fruit_sale['total_weight'].describe()
```

[56]:
```
count     4456.000000
mean      4318.235862
std       3082.321346
min         40.000000
25%       1777.500000
50%       3580.000000
75%       6472.500000
max      20000.000000
Name: total_weight, dtype: float64
```

[57]:
```
fruit_sale_transformed_transaction = pd.read_csv(f"{raw_data}"+␣
↪"fruit_sale_transformed_transaction.csv")

fruit_sale_transformed_transaction.head()
```

[57]:
```
   transaction_id  user_id trans_type  mill_id  do_id  ramp_id  \
0            2272      201       mill      2.0   26.0      NaN
1            2273      201       mill      2.0   26.0      NaN
2            2274      201       mill      2.0   26.0      NaN
3            2275      201       mill      2.0   26.0      NaN
4            2276      201       mill      2.0   26.0      NaN

   ramp_manager_id other_do  grade  bridge  …  added_by  total_weight  \
0              NaN      NaN    0.0     NaN  …     david           NaN
1              NaN      NaN    0.0     NaN  …     david           NaN
2              NaN      NaN    0.0     NaN  …     david           NaN
3              NaN      NaN    0.0     NaN  …     david           NaN
```

```
4              NaN      NaN    0.0     NaN  …     david           NaN

   adjusted_weight      client_time_stamp  app_version  fruit_sale_id  \
0              NaN  2020-11-02 03:00:31.0        2.7.0           3061
1              NaN  2020-11-02 03:00:31.0        2.7.0           3061
2              NaN  2020-11-02 03:00:31.0        2.7.0           3061
3              NaN  2020-11-02 03:00:31.0        2.7.0           3061
4              NaN  2020-11-02 03:00:31.0        2.7.0           3061

   transaction_split_type             deleted_at             created_at  \
0                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
1                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
2                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
3                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0
4                     NaN  2020-11-02 04:00:31.0  2020-11-02 04:00:31.0

             updated_at
0  2020-11-02 04:00:31.0
1  2020-11-02 04:00:31.0
2  2020-11-02 04:00:31.0
3  2020-11-02 04:00:31.0
4  2020-11-02 04:00:31.0

[5 rows x 27 columns]
```

[58]:
```
fruit_sale['fruit_sale_id'] = fruit_sale['fruit_sale_id'].astype(str).
 ↪replace('\.0', '', regex=True)

fruit_sale['date'] = pd.to_datetime(fruit_sale['created_at']).dt.date

fruit_sale['receipt_id'] = ['fs_' + str(row) for row in␣
 ↪fruit_sale['fruit_sale_id']]
```

[59]:
```
#mask = fruit_sale['price'] >2200

#fruit_sale['price'][mask] = np.nan
```

[60]:
```
#mask = fruit_sale['total_weight'] >60000

#fruit_sale['total_weight'][mask] = np.nan
```

[61]:
```
fruit_sale['adjusted_weight'].describe()
```

[61]:
```
count    4371.000000
mean      206.070922
std       202.902695
min        -3.000000
```

```
25%        78.000000
50%       150.000000
75%       290.000000
max      2900.000000
Name: adjusted_weight, dtype: float64
```

[62]:
```
#mask = fruit_sale['adjusted_weight'] <0

#fruit_sale['adjusted_weight'][mask] = np.nan
```

[63]:
```
#mask = fruit_sale['adjusted_weight'] >5000

#fruit_sale['adjusted_weight'][mask] = np.nan
```

[64]:
```
fruit_sale_transformed_transaction['fruit_sale_id'] =␣
↪fruit_sale_transformed_transaction['fruit_sale_id'].astype(str).replace('\.
↪0', '', regex=True)

fruit_sale_transformed_transaction['transaction_id'] =␣
↪fruit_sale_transformed_transaction['transaction_id'].astype(str).replace('\.
↪0', '', regex=True)

fruit_sale_transformed_transaction['date'] = pd.
↪to_datetime(fruit_sale_transformed_transaction['created_at']).dt.date
```

[65]:
```
#print(fruit_sale.shape)

#cols =␣
↪['fruit_sale_id','trans_type','mill_id','ramp_id','do_id','ramp_manager_id']

#fruit_sale = fruit_sale.merge(fruit_sale_transformed_transaction[cols].
↪drop_duplicates(),
#                              on = 'fruit_sale_id',
#                              how = 'left'
#                             )

#print(fruit_sale.shape)
```

[66]:
```
'''
print(fruit_sale.shape)

cols = ['fruit_sale_id','grade']

fruit_sale = fruit_sale.merge(fruit_sale_transformed_transaction[cols].dropna().
↪drop_duplicates(),
                              on = 'fruit_sale_id',
```

```
                                how = 'left'
                               )

print(fruit_sale.shape)
'''
```

[66]:
```
"\nprint(fruit_sale.shape)\n\ncols = ['fruit_sale_id','grade']\n\nfruit_sale = f
ruit_sale.merge(fruit_sale_transformed_transaction[cols].dropna().drop_duplicate
s(), \n                                on = 'fruit_sale_id',\n
how = 'left'\n                                    )\n\nprint(fruit_sale.shape)\n"
```

[67]:
```
#mask = fruit_sale['grade'] >15

#fruit_sale['grade'][mask] = np.nan
```

[68]:
```
fruit_sale.columns
```

[68]:
```
Index(['Unnamed: 0', 'index', 'fruit_sale_id', 'fruit_sale_identifier',
       'telemetry_log_id', 'buyer_type', 'user_id', 'total_weight',
       'total_sale', 'payment_date', 'adjusted_weight', 'bridge',
       'price_fruit_sale', 'cash_available', 'is_shared',
       'created_client_time_stamp', 'app_version', 'updated_client_time_stamp',
       'created_at', 'updated_at', 'deleted_at', 'date', 'receipt_id',
       'price_id', 'trans_type', 'mill_id', 'ramp_id', 'do_id',
       'ramp_manager_id', 'grade_fruit_sale', 'lr_id', 'lrm_id',
       'buyer_receipt', 'buyer', 'special_price_daily_index'],
      dtype='object')
```

[69]:
```
fruit_sale['mill_id'] = fruit_sale['mill_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['lr_id'] = fruit_sale['ramp_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['do_id'] = fruit_sale['do_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale['lrm_id'] = fruit_sale['ramp_manager_id'].astype(str).replace('\.0',␣
↪'', regex=True)

fruit_sale['user_id'] = fruit_sale['user_id'].astype(str).replace('\.0', '',␣
↪regex=True)

fruit_sale.replace('nan',np.nan, inplace=True)
```

[70]:
```
'''
fruit_sale['buyer_receipt'] = (fruit_sale['trans_type'] + '//' +
                               fruit_sale['mill_id'].astype(str) + '//' +
                               'nan' + '//' +
                                fruit_sale['lr_id'].astype(str)
                              )
'''
```

[70]:
```
"\nfruit_sale['buyer_receipt'] = (fruit_sale['trans_type'] + '//' + \n
fruit_sale['mill_id'].astype(str) + '//' +\n                                'nan'
+ '//' +\n                                fruit_sale['lr_id'].astype(str)\n
)\n"
```

[71]:
```
'''
fruit_sale['buyer'] = (fruit_sale['trans_type'] + '//' +
                       fruit_sale['mill_id'].astype(str) + '//' +
                       fruit_sale['do_id'].astype(str) + '//' +
                       fruit_sale['lr_id'].astype(str)
                      )
'''
```

[71]:
```
"\nfruit_sale['buyer'] = (fruit_sale['trans_type'] + '//' + \n
fruit_sale['mill_id'].astype(str) + '//' +\n
fruit_sale['do_id'].astype(str) + '//' +\n
fruit_sale['lr_id'].astype(str)\n                      )\n"
```

[72]:
```
'''
fruit_sale.rename(columns={'grade':'grade_fruit_sale',
                           'price':'price_fruit_sale'
                          }, inplace=True)
'''
```

[72]:
```
"\nfruit_sale.rename(columns={'grade':'grade_fruit_sale',\n
'price':'price_fruit_sale'\n                          }, inplace=True)\n"
```

2.17 Import relevant_village table

[73]:
```
relevant_village = pd.
 ↪read_excel(f"{notebook_2_output}"+"relevant_village_complete.xlsx")

geom_list = []

for geom in relevant_village['geometry']:
    try:
        geom_list.append(shapely.wkt.loads(geom))
    except:
        geom_list.append(np.nan)
```

```
relevant_village['geometry'] = geom_list

#mask = relevant_village['geometry'].notna()

#relevant_village = relevant_village[mask]

#relevant_village.reset_index(inplace = True)

relevant_village.drop(columns=['Unnamed: 0','index'], inplace=True)

relevant_village.head()
```

```
ParseException: Expected word but encountered end of stream
```

[73]:

	level_0	PROVNO	KABKOTNO	KECNO	DESANO	IDSP2010	PROVINSI	KABKOT	KECAMATAN	DESA	SUMBER	geometry	kecamatan_desa	relevant_village_index	nighttime_light	area	nighttime_light_density
0	0	14	4	20	30	1404020030	RIAU	PELALAWAN	PANGKALAN KURAS	PALAS	SP2010	(POLYGON ((101.9055505690001 0.275409893000073…	PANGKALAN KURAS__PALAS	0.0	65.637956	38.116357	1.722042
1	1	14	4	21	10	1404021010	RIAU	PELALAWAN	UKUI	AIR EMAS	SP2010	POLYGON ((102.1206216090001 -0.109291733999953…	UKUI__AIR EMAS	1.0	43.561230	16.328344	2.667829
2	2	14	4	32	11	1404032011	RIAU	PELALAWAN	BANDAR PETALANGAN	AIR TERJUN	SP2010	POLYGON ((102.1467148460001 0.1343687620000651…	BANDAR PETALANGAN__AIR TERJUN	2.0	39.424383	34.678845	1.136842
3	3	14	4	32	6	1404032006	RIAU	PELALAWAN	BANDAR PETALANGAN	ANGKASA	SP2010	POLYGON ((102.125160625 0.189884060000054, 102…	BANDAR PETALANGAN__ANGKASA	3.0	15.253111		
4	4	14	4	30	13	1404030013	RIAU	PELALAWAN	BUNUT	BAGAN LAGUH	SP2010	POLYGON ((102.0921405430001 0.2523691220000615…	BUNUT__BAGAN LAGUH	4.0	141.456076		

```
3   22.252859                 0.685445
4  127.280617                 1.111372
```

[74]:
```
connectedness_village_df = pd.
↪read_excel(f"{notebook_2_output}"+"connectedness_village_df.xlsx")

connectedness_village_df.drop(columns=['Unnamed: 0'],inplace=True)

connectedness_village_df.head()
```

[74]:
```
            buyer_kecamatan_desa          kecamatan_desa  \
0         PANGKALAN KURAS__PALAS  PANGKALAN KURAS__PALAS
1                UKUI__AIR EMAS  PANGKALAN KURAS__PALAS
2  BANDAR PETALANGAN__AIR TERJUN  PANGKALAN KURAS__PALAS
3     BANDAR PETALANGAN__ANGKASA  PANGKALAN KURAS__PALAS
4             BUNUT__BAGAN LAGUH  PANGKALAN KURAS__PALAS

   kecamatan_desa_connected  same_village     user_kecamatan_desa  \
0                         0          True  PANGKALAN KURAS__PALAS
1                         0         False  PANGKALAN KURAS__PALAS
2                         0         False  PANGKALAN KURAS__PALAS
3                         0         False  PANGKALAN KURAS__PALAS
4                         1         False  PANGKALAN KURAS__PALAS

   shortest_path_length
0                   0.0
1                   6.0
2                   5.0
3                   3.0
4                   1.0
```

[ ]:
```

```

[75]:
```
village_neighbors_df = pd.
↪read_excel(f"{notebook_2_output}"+"village_neighbors_df.xlsx")

village_neighbors_df.drop(columns=['Unnamed: 0'],inplace=True)

village_neighbors_df.head()
```

[75]:
```
                kecamatan_desa           user_kecamatan_desa  \
0         PANGKALAN KURAS__PALAS         PANGKALAN KURAS__PALAS
1                UKUI__AIR EMAS                UKUI__AIR EMAS
2  BANDAR PETALANGAN__AIR TERJUN  BANDAR PETALANGAN__AIR TERJUN
3     BANDAR PETALANGAN__ANGKASA     BANDAR PETALANGAN__ANGKASA
4             BUNUT__BAGAN LAGUH             BUNUT__BAGAN LAGUH
```

```
                                village_neighbors_<=5  \
0  ['PANGKALAN KURAS__PALAS', 'BANDAR PETALANGAN_…
1  ['UKUI__AIR EMAS', 'BANDAR PETALANGAN__AIR TER…
2  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
3  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
4  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…

                                village_neighbors_<=8
0  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
1  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
2  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
3  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
4  ['PANGKALAN KURAS__PALAS', 'UKUI__AIR EMAS', '…
```

[76]:
```
village_neighbors_df['village_neighbors_<=5'] = [
    a.replace("\"","").replace("[","").replace(", ",",").replace("'","").
 ↪replace("' ","'").split(",")
    for a in village_neighbors_df['village_neighbors_<=5']
]
```

[77]:
```
village_neighbors_df['village_neighbors_<=8'] = [
    a.replace("\"","").replace("[","").replace(", ",",").replace("'","").
 ↪replace("' ","'").split(",")
    for a in village_neighbors_df['village_neighbors_<=8']
]
```

3 Creating relevant tables

3.1 Create users_receipt

3.1.1 Extract row from receipt table

[78]:
```
def get_same_day_price_receipt(data, row):
    date = data['date'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]
    user_id = data['user_id'].loc[row]

    mask = ((data['date'] == date) &
            (data['buyer_receipt'] == buyer_receipt) &
            (data['user_id']==user_id) &
            ~(data['price_receipt'].isna())
           )

    tmp_df = data[mask]

    if tmp_df.empty:
        output = np.nan
```

```
    else :
        output = list(tmp_df['receipt_id'])

    return output
```

[79]:
```
def get_same_day_weight_receipt(data, row):
    #print(row)
    date = data['date'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]
    user_id = data['user_id'].loc[row]

    mask = ((data['date'] == date) &
            (data['buyer_receipt'] == buyer_receipt) &
            (data['user_id']==user_id) &
            ~(data['weight_receipt'].isna())
           )

    tmp_df = data[mask]

    if tmp_df.empty:
        output = np.nan
    else :
        output = list(tmp_df['receipt_id'])

    return output
```

[80]:
```
transactions_sql['net_weight'].max()
```

[80]:
```
992000.0
```

[81]:
```
receipt['same_day_weight_receipts'] = np.nan
```

[82]:
```
receipt['same_day_weight_receipts'] = receipt['same_day_weight_receipts'].
↪astype(object)

receipt['same_day_weight_receipts'].dtypes
```

[82]:
```
dtype('O')
```

[83]:
```
mask = ~(receipt['price_receipt'].isna())

for row in receipt[mask].index:
    receipt.at[row,'same_day_weight_receipts'] =␣
↪get_same_day_weight_receipt(receipt, row)
```

[84]:
```
receipt['same_day_price_receipts'] = np.nan
```

```
receipt['same_day_price_receipts'] = receipt['same_day_price_receipts'].
↪astype(object)

mask = ~(receipt['weight_receipt'].isna())

for row in receipt[mask].index:
    receipt.at[row,'same_day_price_receipts'] =␣
↪get_same_day_price_receipt(receipt, row)
```

[85]:
```
receipt['price_|_weight_no_price'] = False

mask = (~(receipt['price_receipt'].isna()) | ## Including all the price receipt
        (~(receipt['weight_receipt'].isna()) &          ## Including all the␣
↪weight receipt withou price receipt
         (receipt['same_day_price_receipts'].isna())
        )
       )

receipt['price_|_weight_no_price'][mask] = True
```

[86]:
```
mask = ((receipt['user_id'].isin(users_analyzed)) &
        (receipt['price_|_weight_no_price'])
       )

users_receipt = receipt[mask]

users_receipt['data_source'] ='receipt'

users_receipt.shape
```

[86]: (5221, 25)

3.1.2 Merging fruit_sale table

[87]:
```
transactions_sql['transaction_id'] = transactions_sql['id']
```

[88]:
```
###duplicate_fruit_sale_id = [5561]

mask = fruit_sale['user_id'].isin(users_analyzed)

fruit_sale = fruit_sale[mask]

print(fruit_sale.shape)
```

```
(3350, 35)
```

[89]:
```
mask = ~fruit_sale['buyer_receipt'].isna()

fruit_sale = fruit_sale[mask]

print(fruit_sale.shape)
```

```
(2366, 35)
```

[90]:
```
fruit_sale['data_source'] = 'fruit_sale'
```

[91]:
```
print(users_receipt.shape)

users_receipt = pd.concat([users_receipt, fruit_sale], sort=False)

print(users_receipt.shape)
```

```
(5221, 25)
(7587, 47)
```

3.1.3 Extract row from transactions_sql table

[92]:
```
cols = ['user_id','date','buyer_receipt','receipt_id']

mergee = receipt[cols].
 ↪drop_duplicates(subset=['user_id','date','buyer_receipt'], keep='first')

print(transactions_sql.shape)

transactions_sql = transactions_sql.merge(mergee,
                                          on =␣
 ↪['user_id','date','buyer_receipt'],
                                          how = 'left'
                                         )

print(transactions_sql.shape)
```

```
(88253, 82)
(88253, 83)
```

[93]:
```
mask = ((transactions_sql['receipt_id'].isna()) &
        (transactions_sql['user_id'].isin(users_analyzed)) &
        ((transactions_sql['role'].isin(['Middle Man', 'Amprah Farmer', 'Do/
 ↪Middle Man','Farmer'])) |
         ((transactions_sql['role']=='Ramp Manager')␣
 ↪&(transactions_sql['trans_type']=='mill')
         )
        )
```

```
      )

cols = list(set(users_receipt.columns) & set(transactions_sql.columns))

transactions_no_receipt = (transactions_sql[cols][mask].
                           ␣
 ↪drop_duplicates(subset=['user_id','date','buyer_receipt'],
                                          keep= 'first')
                           )

transactions_no_receipt.shape
```

[93]:
```
(419, 22)
```

[94]:
```
transactions_no_receipt['data_source'] = 'transactions_sql'

#users_receipt['data_source'] ='receipt'
```

[95]:
```
for column in users_receipt.columns:
    if column in transactions_no_receipt.columns:
        transactions_no_receipt[column] = transactions_no_receipt[column]
    else :
        transactions_no_receipt[column] = np.nan
```

[96]:
```
transactions_no_receipt['receipt_id'] = ['trx_' + str(row) for row in␣
 ↪transactions_no_receipt.index]
```

[97]:
```
print(users_receipt.shape)

users_receipt = pd.concat([users_receipt,transactions_no_receipt],
                          ignore_index=True
                         )

print(users_receipt.shape)
```

```
(7587, 47)
(8006, 47)
```

3.1.4 Filtering duplicates

[98]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(users_receipt_duplicates,
                                    how='left',
                                    on=['receipt_id',
                                        'date',
                                        'buyer_receipt'
```

```
                                        ],
                                       indicator=True
                                      )

print(users_receipt.shape)
```

```
(8006, 47)
(8006, 58)
```

[99]:
```
mask = ~(users_receipt['duplicates_final'].isin(['Duplicate']))

users_receipt = users_receipt[mask]

users_receipt.shape
```

[99]:
```
(7933, 58)
```

3.1.5 Merging columns from users & relevant village table

[100]:
```
cols = ['user_id','role','kecamatan','kecamatan_desa']

print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols],
                                    on= 'user_id',
                                    how='left'
                                   )

print(users_receipt.shape)
```

```
(7933, 58)
(7933, 61)
```

[101]:
```
'''cols = ['geometry','kecamatan_desa']

print(users_receipt.shape)

users_receipt = users_receipt.merge(relevant_village[cols],
                                    on = 'kecamatan_desa',
                                    how = 'left'
                                   )

print(users_receipt.shape)
'''
```

[101]:
```
"cols =
['geometry','kecamatan_desa']\n\nprint(users_receipt.shape)\n\nusers_receipt =
```

```
users_receipt.merge(relevant_village[cols],\n
on = 'kecamatan_desa',\n                                    how = 'left'\n
)\n\nprint(users_receipt.shape)\n"
```

3.1.6 Impute buyer information for transaction from receipt table

[102]:
```
def get_corresponding_transactions_sql_row(data,row):
    date = data['date'].loc[row]
    user_id = data['user_id'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    mask = ((transactions_sql['date']==date) &
            (transactions_sql['user_id']==user_id) &
            (transactions_sql['buyer_receipt']==buyer_receipt) &
            (transactions_sql['price_sharing']) &
            ~(transactions_sql['outlier'])
           )

    tmp_df = transactions_sql[mask]

    if tmp_df.empty:
        output = np.nan
    else :
        index = tmp_df['price_cleaned'].idxmax()
        output = index
    return output
```

[103]:
```
transactions_sql['date'] = pd.to_datetime(transactions_sql['date']).dt.date
```

[104]:
```
users_receipt['corresponding_transactions_sql_row'] = np.nan

users_receipt['corresponding_transactions_sql_row'] = [
    get_corresponding_transactions_sql_row(users_receipt,row)
    for row in users_receipt.index
    ]
```

[105]:
```
users_receipt['buyer'].isna().sum()
```

[105]:
```
5148
```

[106]:
```
users_receipt['corresponding_transactions_sql_row']
```

[106]:
```
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
```

```
        …
7928        NaN
7929        NaN
7930        NaN
7931    86252.0
7932        NaN
Name: corresponding_transactions_sql_row, Length: 7933, dtype: float64
```

[107]:
```
#users_receipt['buyer'] = np.nan

#mask = (users_receipt['corresponding_transactions_sql_row'].notna())

users_receipt['buyer_from_transactions_sql'] = [
    transactions_sql['buyer'].
 ↪loc[users_receipt['corresponding_transactions_sql_row'].loc[index]]
    if pd.notna(users_receipt['corresponding_transactions_sql_row'].loc[index])
    else np.nan for index in users_receipt.index
    ]
```

[108]:
```
users_receipt['buyer_receipt_from_transactions_sql'] = [
    transactions_sql['buyer_receipt'].
 ↪loc[users_receipt['corresponding_transactions_sql_row'].loc[index]]
    if pd.notna(users_receipt['corresponding_transactions_sql_row'].loc[index])
    else np.nan for index in users_receipt.index
    ]
```

[109]:
```
mask = (users_receipt['corresponding_transactions_sql_row'].notna())

(users_receipt['buyer_receipt_from_transactions_sql'][mask]␣
 ↪==users_receipt['buyer_receipt'][mask]).describe()
```

[109]:
```
count     4362
unique       1
top       True
freq      4362
dtype: object
```

[110]:
```
mask = (users_receipt['buyer'].isna())

users_receipt['data_source'][mask].value_counts()
```

[110]:
```
receipt    5148
Name: data_source, dtype: int64
```

[111]:
```
users_receipt['buyer'].fillna(users_receipt['buyer_from_transactions_sql'],␣
 ↪inplace=True)
```

[112]:
```
users_receipt['buyer'].fillna(users_receipt['buyer_receipt'], inplace=True)
```

[113]:
```
mask = (users_receipt['trans_type']=='lr')

(users_receipt['buyer'][mask]==users_receipt['buyer_receipt'][mask]).describe()
```

[113]:
```
count     3196
unique       1
top       True
freq      3196
dtype: object
```

[114]:
```
mask = ((users_receipt['trans_type']=='lr')&
        (users_receipt['buyer']!=users_receipt['buyer_receipt']))

users_receipt[['buyer','buyer_receipt']][mask]
```

[114]:
```
Empty DataFrame
Columns: [buyer, buyer_receipt]
Index: []
```

[115]:
```
users_receipt['buyer_receipt'].loc[0][:8]
```

[115]:
```
'mill//2/'
```

[116]:
```
users_receipt['buyer_receipt_:8'] = [row[:8] for row in␣
↪users_receipt['buyer_receipt']]
```

[117]:
```
users_receipt['buyer_:8'] = [row[:8] for row in users_receipt['buyer']]
```

[118]:
```
users_receipt['buyer']
```

[118]:
```
0        mill//2//nan//nan
1        mill//2//nan//nan
2        mill//2//nan//nan
3        mill//2//nan//nan
4        mill//1//nan//nan
               …
7928      mill//5//52//nan
7929    mill//98//205//nan
7930     mill//4//nan//nan
7931     mill//17//45//nan
7932      mill//5//52//nan
Name: buyer, Length: 7933, dtype: object
```

[119]:
```
mask = (users_receipt['trans_type']=='mill')
```

```
(users_receipt['buyer_:8'][mask]==users_receipt['buyer_receipt_:8'][mask]).
↪describe()
```

[119]:
```
count     4737
unique       1
top       True
freq      4737
dtype: object
```

3.1.7 Calculate price for each receipt

[120]:
```
users_receipt['mill_id'] = users_receipt['mill_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_receipt['lr_id'] = users_receipt['ramp_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_receipt['do_id'] = users_receipt['do_id'].astype(str).replace('\.0', '',␣
↪regex=True)

users_receipt['user_id'] = users_receipt['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)

users_receipt.replace('nan',np.nan, inplace = True)
```

[121]:
```
def get_special_price(data,row):
    date = data['date'].loc[row]
    user_id = data['user_id'].loc[row]
    buyer = data['buyer'].loc[row]

    mask = ((special_price_daily['date']==date) &
            (special_price_daily['user_id']==user_id) &
            (special_price_daily['buyer']==buyer)
           )

    tmp = special_price_daily[mask]

    output = np.nan

    if tmp.empty:
        output= np.nan
    else :
        output= np.nanmax(tmp['price'])

    return output
```

[122]:
```
users_receipt['date'] = pd.to_datetime(users_receipt['date']).dt.date
```

[123]:
```
users_receipt['special_price'] = [get_special_price(users_receipt,row) for
                                  row in users_receipt.index
                                 ]
```

[124]:
```
users_receipt['special_price_transaction'] = False

mask = ~users_receipt['special_price'].isna()

users_receipt['special_price_transaction'][mask] = True
```

[125]:
```
def get_special_price_buyer_trx(row):
    data = users_receipt
    user_id = data['user_id'].loc[row]
    buyer = data['buyer'].loc[row]

    mask = ((price_group_user['user_id']==user_id) &
            (price_group_user['buyer']==buyer)
           )

    tmp = price_group_user[mask]

    output = len(tmp) > 0

    return output
```

[126]:
```
get_special_price_buyer_trx(0)
```

[126]:
```
False
```

[127]:
```
users_receipt['special_price_buyer_trx'] = [get_special_price_buyer_trx(row) for
                                            row in users_receipt.index
                                           ]
```

[128]:
```
users_receipt['price_cleaned'] = np.nan

#mask = ~(users_receipt['corresponding_transactions_sql_row'].isna())

users_receipt['price_cleaned'] = [
    transactions_sql['price_cleaned'].
 ↪loc[users_receipt['corresponding_transactions_sql_row'].loc[index]]
    if pd.notna(users_receipt['corresponding_transactions_sql_row'].loc[index])
    else np.nan for index in users_receipt.index
    ]
```

3.1.8 Price imputation

[129]:
```
def impute_price_counterfactual_price_table(data, row):
    #print(row)
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    mask_2 = ((counterfactual_price['date']==date) &
              #(counterfactual_price['user_id']==user_id) &
              (counterfactual_price['buyer_receipt']==buyer_receipt) #&
              #(transactions_sql['price_sharing']) &
              #~(transactions_sql['id'].isin(outlier_id))
             )

    tmp_df = counterfactual_price[mask_2]

    if tmp_df.empty:
        output = np.nan
    else:
        output = np.nanmax(tmp_df['price_counterfactual_table'])

    return output
```

[130]:
```
users_receipt['imputed_price_counterfactual_table'] = np.nan

mask = ((users_receipt['price_cleaned'].isna())
       )

users_receipt['imputed_price_counterfactual_table'] = [
    impute_price_counterfactual_price_table(users_receipt, row)
    for row in users_receipt.index
]
```

```
C:\Users\AK\Anaconda3\lib\site-packages\ipykernel_launcher.py:19:
RuntimeWarning: All-NaN axis encountered
```

[131]:
```
counterfactual_price_manual = pd.pivot_table(users_receipt_duplicates,
                                             index = ['date','buyer_receipt'],
                                             values = 'imputed_price_manual',
                                             aggfunc = np.nanmedian
                                            )

counterfactual_price_manual.columns =␣
↪['imputed_price_manual_counterfactual_table']

counterfactual_price_manual.reset_index(inplace =True)
```

```
counterfactual_price_manual['date'] = pd.
 ↪to_datetime(counterfactual_price_manual['date']).dt.date
```

[132]:
```
print(users_receipt.shape)

#cols = ['date','buyer_receipt','imputed_price_manual_counterfactual_table']

users_receipt = users_receipt.merge(counterfactual_price_manual,
                                    on = ['date','buyer_receipt'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 71)
(7933, 72)
```

[133]:
```
users_receipt['price_imputed_final'] = users_receipt['imputed_price_manual']

users_receipt['price_imputed_final'].dropna().shape
```

[133]:
```
(1213,)
```

[134]:
```
users_receipt['price_source'] = np.nan

mask = ~users_receipt['price_imputed_final'].isna()

users_receipt['price_source'][mask] = 'Manual Imputation'
```

[135]:
```
users_receipt['price_imputed_final'].fillna(users_receipt['special_price'],␣
 ↪inplace = True)
```

[136]:
```
mask = ((users_receipt['special_price'].notna()) &
        (users_receipt['imputed_price_manual'].isna())
       )

users_receipt['price_source'][mask] = 'special_price_daily'
```

[137]:
```
users_receipt['price_imputed_final'].fillna(users_receipt['price_fruit_sale'],␣
 ↪inplace = True)
```

[138]:
```
mask = ((~users_receipt['price_fruit_sale'].isna()) &
        (users_receipt['special_price'].isna()) &
        (users_receipt['imputed_price_manual'].isna())
       )
```

```
users_receipt['price_source'][mask] = 'price fruit sale'
```

[139]:
```
users_receipt['price_imputed_final'].fillna(users_receipt['price_cleaned'],␣
↪inplace = True)
```

[140]:
```
mask = ((~users_receipt['price_cleaned'].isna()) &
        (users_receipt['price_fruit_sale'].isna()) &
        (users_receipt['special_price'].isna()) &
        (users_receipt['imputed_price_manual'].isna())
       )

users_receipt['price_source'][mask] = 'Transactions_sql'
```

[141]:
```
users_receipt['price_imputed_final'].
↪fillna(users_receipt['imputed_price_counterfactual_table'],inplace = True)
```

[142]:
```
mask = ((~users_receipt['imputed_price_counterfactual_table'].isna()) &
        (users_receipt['price_cleaned'].isna()) &
        (users_receipt['price_fruit_sale'].isna()) &
        (users_receipt['special_price'].isna()) &
        (users_receipt['imputed_price_manual'].isna())
       )

users_receipt['price_source'][mask] = 'Counterfactual price table'
```

[143]:
```
users_receipt['price_source'].value_counts()
```

[143]:
```
Transactions_sql              3327
price fruit sale              2194
Manual Imputation             1213
Counterfactual price table     739
special_price_daily            276
Name: price_source, dtype: int64
```

3.1.9 Calculate grade and Post_Grading_Price

[144]:
```
cols =␣
↪['date','buyer_receipt','grade_counterfactual_table','buyer','post_grading_price_counterfac

print(users_receipt.shape)

users_receipt = users_receipt.merge(counterfactual_price[cols],
                                    on = ['date','buyer_receipt','buyer'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 74)
(7933, 76)
```

[145]:
```
#users_receipt.rename(columns = {'post_grading_price':
↪'post_grading_price_buyer'}, inplace = True)
```

[146]:
```
cols = ['date','buyer_receipt','grade_cleaned_buyer_receipt']

print(users_receipt.shape)

users_receipt = users_receipt.merge(counterfactual_price_buyer_receipt[cols],
                                    on = ['date','buyer_receipt'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 76)
(7933, 77)
```

[147]:
```
users_receipt['grade_imputed_final'] = users_receipt['imputed_grade_manual']
```

[148]:
```
users_receipt['grade_source'] = np.nan

mask = ~users_receipt['grade_imputed_final'].isna()

users_receipt['grade_source'][mask] = 'Manual imputation'
```

[149]:
```
users_receipt['grade_imputed_final'].fillna(users_receipt['grade_fruit_sale'],␣
↪inplace = True)
```

[150]:
```
mask = ((~users_receipt['grade_fruit_sale'].isna()) &
        (users_receipt['imputed_grade_manual'].isna())
       )
```

```
users_receipt['grade_source'][mask] = 'fruit_sale'
```

[151]:
```
users_receipt['grade_imputed_final'].
 ↪fillna(users_receipt['grade_counterfactual_table'], inplace = True)
```

[152]:
```
mask = ((~users_receipt['grade_counterfactual_table'].isna()) &
        (users_receipt['grade_fruit_sale'].isna()) &
        (users_receipt['imputed_grade_manual'].isna())
       )

users_receipt['grade_source'][mask] = 'Counterfactual price table'
```

[153]:
```
users_receipt['grade_imputed_final'].
 ↪fillna(users_receipt['grade_cleaned_buyer_receipt'], inplace = True)
```

[154]:
```
users_receipt['grade_imputed_final'].dropna().shape
```

[154]:
```
(7585,)
```

[155]:
```
mask = ((users_receipt['grade_cleaned_buyer_receipt'].notna()) &
        (users_receipt['grade_counterfactual_table'].isna()) &
        (users_receipt['grade_fruit_sale'].isna()) &
        (users_receipt['imputed_grade_manual'].isna())
       )

users_receipt['grade_source'][mask] = 'Counterfactual price table buyer receipt'
```

[156]:
```
users_receipt['grade_fruit_sale_calculated'] = users_receipt['adjusted_weight']/
 ↪users_receipt['total_weight']*100
```

[157]:
```
mask = users_receipt['grade_fruit_sale_calculated']<=1

users_receipt['grade_fruit_sale_calculated'][mask] = np.nan
```

[158]:
```
mask = users_receipt['grade_fruit_sale_calculated']>=15

users_receipt['grade_fruit_sale_calculated'][mask] = np.nan
```

[159]:
```
users_receipt['grade_fruit_sale_calculated'].describe()
```

[159]:
```
count    2245.000000
mean        4.421722
std         1.041712
min         1.250000
25%         3.980583
50%         4.483395
75%         4.921466
```

```
max        14.814815
Name: grade_fruit_sale_calculated, dtype: float64
```

[160]:
```
users_receipt['grade_imputed_final'].
↪fillna(users_receipt['grade_fruit_sale_calculated'], inplace = True)
```

[161]:
```
users_receipt['grade_imputed_final'].dropna().shape
```

[161]:
```
(7585,)
```

[162]:
```
mask = ((users_receipt['grade_fruit_sale_calculated'].notna()) &
        (users_receipt['grade_cleaned_buyer_receipt'].isna()) &
        (users_receipt['grade_counterfactual_table'].isna()) &
        (users_receipt['grade_fruit_sale'].isna()) &
        (users_receipt['imputed_grade_manual'].isna())
       )

users_receipt['grade_source'][mask] = 'fruit_sale_calculated'
```

[163]:
```
mask = users_receipt['grade_imputed_final']<1

users_receipt['grade_imputed_final'][mask] = np.nan
```

[164]:
```
users_receipt['grade_imputed_final'].describe()
```

[164]:
```
count    7585.000000
mean        4.574641
std         1.101844
min         1.000000
25%         4.000000
50%         4.490000
75%         4.990000
max        14.810000
Name: grade_imputed_final, dtype: float64
```

[165]:
```
users_receipt['post_grading_price'] = (users_receipt['price_imputed_final']*
                                       (1-users_receipt['grade_imputed_final']/
↪100)
                                      )
```

[166]:
```
users_receipt['post_grading_price_source'] = np.nan

mask = users_receipt['post_grading_price'].notna()

users_receipt['post_grading_price_source'][mask] = 'individual report'
```

[167]:
```
users_receipt['post_grading_price'].
↪fillna(users_receipt['post_grading_price_counterfactual_table'], inplace =␣
↪True)
```

[168]:
```
mask = ((users_receipt['post_grading_price'].isna())&
         (users_receipt['post_grading_price_counterfactual_table'].notna())
        )

users_receipt['post_grading_price_source'][mask] = 'Counterfactual price table'
```

[169]:
```
users_receipt['post_grading_price_source'].value_counts()
```

[169]:
```
individual report    7481
Name: post_grading_price_source, dtype: int64
```

[170]:
```
print(users_receipt.shape)

cols = ['buyer','buyer_kecamatan_desa']

users_receipt = users_receipt.merge(buyers[cols],
                                    on = 'buyer',
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 82)
(7933, 83)
```

[171]:
```
users_receipt['buyer_in_the_same_village'] = (users_receipt['kecamatan_desa'] ==
                                              ␣
↪users_receipt['buyer_kecamatan_desa']
                                              )
```

3.1.10 Create old_buyers_df table

[172]:
```
cols = ['user_id','buyer','buyer_receipt','source']

old_buyers_df = pd.concat([users_do_ramp[cols],special_price_daily[cols]]).
↪dropna().drop_duplicates(keep ='first')
```

[173]:
```
old_buyers_df['trans_type'] = [row[:row.index('//')] for row in␣
↪old_buyers_df['buyer']]
```

3.1.11 Calculate counterfactual price and buyer

[174]:
```
'''
users_receipt_price = pd.pivot_table(users_receipt,
                                     index=['buyer','date'],
                                     values =␣
 ↪['price_imputed_final','post_grading_price'],
                                     aggfunc= [np.nanmax, np.nanmedian]
                                     )#.reset_index()

#users_receipt_price.rename(columns= {'post_grading_price':
 ↪'max_post_grading_price_users_receipt',
#                                     'price_imputed_final':
 ↪'max_price_imputed_final_users_receipt'},
#                                  inplace = True
#                                  )

users_receipt_price.columns = [users_receipt_price.columns[col][0] + '_' +␣
 ↪users_receipt_price.columns[col][1]
                               + '_users_receipt'
                               for col in range(len(users_receipt_price.
 ↪columns))]

users_receipt_price.reset_index(inplace=True)

users_receipt_price
'''
```

[174]:
```
"\nusers_receipt_price = pd.pivot_table(users_receipt, \n
index=['buyer','date'],\n                                                values =
['price_imputed_final','post_grading_price'],\n
aggfunc= [np.nanmax, np.nanmedian]\n
)#.reset_index()\n\n#users_receipt_price.rename(columns=
{'post_grading_price':'max_post_grading_price_users_receipt',\n#
'price_imputed_final':'max_price_imputed_final_users_receipt'},\n#
inplace = True\n#
)\n\nusers_receipt_price.columns = [users_receipt_price.columns[col][0] + '_' +
users_receipt_price.columns[col][1] \n                                   +
'_users_receipt'\n                                  for col in range(len(users_rece
ipt_price.columns))]\n\nusers_receipt_price.reset_index(inplace=True)\n\nusers_r
eceipt_price\n"
```

[175]:
```
'''
print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(users_receipt_price,
                                                  on = ['buyer','date'],
```

```
                                                    how = 'left'
                                                   )

print(counterfactual_price.shape)
'''
```

[175]:
```
"\nprint(counterfactual_price.shape)\n\ncounterfactual_price =
counterfactual_price.merge(users_receipt_price, \n
on = ['buyer','date'],\n                                                    how =
'left'\n
)\n\nprint(counterfactual_price.shape)\n"
```

[176]:
```
'''
users_receipt['buyer_receipt'] = (users_receipt['trans_type'] + '//' +
                                  users_receipt['mill_id'].astype(str) + '//
 ↪' +
                                  'nan' + '//' +
                                  users_receipt['lr_id'].astype(str)
                                 )
'''
```

[176]:
```
"\nusers_receipt['buyer_receipt'] = (users_receipt['trans_type'] + '//' + \n
users_receipt['mill_id'].astype(str) + '//' +\n
'nan' + '//' +\n
users_receipt['lr_id'].astype(str)\n                                        )\n"
```

[177]:
```
'''
users_receipt['buyer'] = (users_receipt['trans_type'] + '//' +
                          users_receipt['mill_id'].astype(str) + '//' +
                            users_receipt['do_id'].astype(str) + '//' +
                           users_receipt['lr_id'].astype(str)
                          )
'''
```

[177]:
```
"\nusers_receipt['buyer'] = (users_receipt['trans_type'] + '//' + \n
users_receipt['mill_id'].astype(str) + '//' +\n
users_receipt['do_id'].astype(str) + '//' +\n
users_receipt['lr_id'].astype(str)\n                          )\n"
```

[178]:
```
'''
counterfactual_price['post_grading_price'].
 ↪fillna(counterfactual_price['nanmedian_post_grading_price_users_receipt'],
                                                 inplace=True)

counterfactual_price['price_imputed_final'].
 ↪fillna(counterfactual_price['nanmedian_price_imputed_final_users_receipt'],
                                                  inplace=True)
```

```
'''
```

[178]:
```
"\ncounterfactual_price['post_grading_price'].fillna(counterfactual_price['nanme
dian_post_grading_price_users_receipt'], \n
inplace=True)\n\ncounterfactual_price['price_imputed_final'].fillna(counterfactu
al_price['nanmedian_price_imputed_final_users_receipt'], \n
inplace=True)\n"
```

[179]:
```
mask = users_receipt['buyer_receipt']=='lr//nan//nan//nan'

users_receipt['buyer_receipt'][mask]
```

[179]:
```
Series([], Name: buyer_receipt, dtype: object)
```

[180]:
```
mask = users_receipt['buyer_receipt']=='mill//nan//nan//nan'

users_receipt['buyer_receipt'][mask]
```

[180]:
```
Series([], Name: buyer_receipt, dtype: object)
```

[181]:
```
def check_new_buyer_receipt_row(data,row):
    user_id = data['user_id'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_receipt_list = list(old_buyers_df['buyer_receipt'].loc[mask])
    trans_type_list = list(old_buyers_df['trans_type'].loc[mask].unique())
    buyer_receipt = data['buyer_receipt'].loc[row]
    if isnan(data['buyer_receipt'].loc[row]):
        output = np.nan
        case = np.nan
    ### check for special case where buyer_receipt = 'lr//nan//nan//nan' and␣
↪there is ramp in old buyers
    elif ((buyer_receipt =='lr//nan//nan//nan')&('lr' in trans_type_list)):
        output = False
        case = 'False, No ramp id, there is ramp in old buyer_receipt_list'
    ### check for special case where buyer_receipt = 'lr//nan//nan//nan' and no␣
↪ramp in old buyers
    elif ((buyer_receipt =='lr//nan//nan//nan')&('lr' not in trans_type_list)):
        output = True
        case = 'True, No ramp id, no ramp in old buyer_receipt_list'
    ### check for special case where buyer_receipt = 'mill//nan//nan//nan' and␣
↪there is mill in old buyers
    elif ((buyer_receipt =='mill//nan//nan//nan')&('mill' in trans_type_list)):
        output = False
        case = 'False, No mill id, there is mill in old buyer_receipt_list'
```

```
    ### check for special case where buyer_receipt = 'mill//nan//nan//nan' and␣
↪no mill in old buyers
    elif ((buyer_receipt =='mill//nan//nan//nan')&('mill' not in␣
↪trans_type_list)):
        output = True
        case = 'True, No ramp id, no ramp in old buyer_receipt_list'
    elif buyer_receipt in buyer_receipt_list:
        output = False
        case = 'False, buyer_receipt in buyer_receipt_list'
    else :
        output = True
        case = 'True, not edge case'
    return (output,case)
```

[182]:
```
def check_new_buyer_receipt_robust_special_price_row(data,row):
    user_id = data['user_id'].loc[row]
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_receipt_list = list(old_buyers_df['buyer_receipt'].loc[mask])
    trans_type_list = list(old_buyers_df['trans_type'].loc[mask].unique())
    buyer_receipt = data['buyer_receipt'].loc[row]
    if isnan(data['buyer_receipt'].loc[row]):
        output = np.nan
        case = np.nan
    ### check for special case where buyer_receipt = 'lr//nan//nan//nan' and␣
↪there is ramp in old buyers
    elif ((buyer_receipt =='lr//nan//nan//nan')&('lr' in trans_type_list)):
        output = False
        case = 'False, No ramp id, there is ramp in old buyer_receipt_list'
    ### check for special case where buyer_receipt = 'lr//nan//nan//nan' and no␣
↪ramp in old buyers
    elif ((buyer_receipt =='lr//nan//nan//nan')&('lr' not in trans_type_list)):
        output = True
        case = 'True, No ramp id, no ramp in old buyer_receipt_list'
    ### check for special case where buyer_receipt = 'mill//nan//nan//nan' and␣
↪there is mill in old buyers
    elif ((buyer_receipt =='mill//nan//nan//nan')&('mill' in trans_type_list)):
        output = False
        case = 'False, No mill id, there is mill in old buyer_receipt_list'
    ### check for special case where buyer_receipt = 'mill//nan//nan//nan' and␣
↪no mill in old buyers
    elif ((buyer_receipt =='mill//nan//nan//nan')&('mill' not in␣
↪trans_type_list)):
        output = True
        case = 'True, No ramp id, no ramp in old buyer_receipt_list'
```

```
    elif buyer_receipt in buyer_receipt_list:
        output = False
        case = 'False, buyer_receipt in buyer_receipt_list'
    else :
        output = True
        case = 'True, not edge case'
    return (output,case)
```

[183]:
```
check_new_buyer_receipt_row(users_receipt,0)
```

[183]:
```
(False, 'False, buyer_receipt in buyer_receipt_list')
```

[184]:
```
mask = ((old_buyers_df['user_id']=='35') #&
            #(old_buyers_df['source']!= 'special price membership')
           )

old_buyers_df.loc[mask]
```

[184]:

	user_id	buyer	buyer_receipt		source	trans_type
1492	35	mill//1//nan//nan	mill//1//nan//nan	2018	survey	mill
1493	35	mill//2//nan//nan	mill//2//nan//nan	2018	survey	mill

[185]:
```
users_receipt['new_buyer_receipt'] = np.nan

users_receipt['new_buyer_receipt'] =␣
↪[check_new_buyer_receipt_row(users_receipt,row)
                                    for row in users_receipt.index
                                   ]
```

[186]:
```
users_receipt['new_buyer_receipt_robust_special_group'] = np.nan

users_receipt['new_buyer_receipt_robust_special_group'] =␣
↪[check_new_buyer_receipt_robust_special_price_row(users_receipt,row)
                                                        for row in␣
↪users_receipt.index
                                                       ]
```

[187]:
```
def check_new_buyer(data,row):
    user_id = data['user_id'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])
    new_buyer_receipt = data['new_buyer_receipt'].loc[row][0]
    trans_type = data['trans_type'].loc[row]
```

```
    ## do_id does not exist if trans_type == 'mill' and do_id_marker =='/nan//
↪nan'
    do_id_marker = data['buyer'].loc[row][-9:]
    mill_id = data['mill_id'].loc[row]
    buyer = data['buyer'].loc[row]

    if isnan(data['buyer'].loc[row]):
        output = np.nan
        case = 'NaN, no buyer information'
    elif new_buyer_receipt==True:
        output = True
        if trans_type == 'mill':
            case = 'True, new mill transaction'
        elif trans_type == 'lr':
            case = 'True, new ramp transaction'
    ## check for special case where trans_type = 'mill', buyer == 'mill//nan//
↪nan//nan',
    ### and not transacting in a new mill
    elif ((trans_type =='mill') & (buyer == 'mill//nan//nan//nan')&␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, mill transaction mill unknown'
    ### check for special case where trans_type = 'mill', do_id_marker =='/nan//
↪nan',
    ### and not transacting in a new mill
    elif ((trans_type =='mill') & (do_id_marker == '/nan//nan') &␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, mill transaction no do_id'
    ## check for special case where trans_type = 'lr', buyer == 'lr//nan//nan//
↪nan',
    ### and not transacting in a new ramp
    elif ((trans_type =='lr') & (buyer == 'lr//nan//nan//nan')&␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, lr transaction ramp unknown'
    elif buyer in buyer_list:
        output = False
        case = 'False, buyer in old buyer list'
    else :
        output = True
        case = 'True, not edge case'
    return (output,case)
```

[188]:
```
def check_new_buyer_robust_special_group(data,row):
    user_id = data['user_id'].loc[row]
```

```
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])
    new_buyer_receipt = data['new_buyer_receipt_robust_special_group'].
↪loc[row][0]
    trans_type = data['trans_type'].loc[row]
    ## do_id does not exist if trans_type == 'mill' and do_id_marker =='/nan//
↪nan'
    do_id_marker = data['buyer'].loc[row][-9:]
    mill_id = data['mill_id'].loc[row]
    buyer = data['buyer'].loc[row]

    if isnan(data['buyer'].loc[row]):
        output = np.nan
        case = 'NaN, no buyer information'
    elif new_buyer_receipt==True:
        output = True
        if trans_type == 'mill':
            case = 'True, new mill transaction'
        elif trans_type == 'lr':
            case = 'True, new ramp transaction'
    ## check for special case where trans_type = 'mill', buyer == 'mill//nan//
↪nan//nan',
    ### and not transacting in a new mill
    elif ((trans_type =='mill') & (buyer == 'mill//nan//nan//nan')&␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, mill transaction mill unknown'
    ### check for special case where trans_type = 'mill', do_id_marker =='/nan//
↪nan',
    ### and not transacting in a new mill
    elif ((trans_type =='mill') & (do_id_marker == '/nan//nan') &␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, mill transaction no do_id'
    ## check for special case where trans_type = 'lr', buyer == 'lr//nan//nan//
↪nan',
    ### and not transacting in a new ramp
    elif ((trans_type =='lr') & (buyer == 'lr//nan//nan//nan')&␣
↪(new_buyer_receipt==False)):
        output = False
        case = 'False, lr transaction ramp unknown'
    elif buyer in buyer_list:
        output = False
        case = 'False, buyer in old buyer list'
```

```
    else :
        output = True
        case = 'True, not edge case'
    return (output,case)
```

[189]:
```
users_receipt[['buyer','buyer_receipt']].head(13)
```

[189]:

	buyer	buyer_receipt
0	mill//2//nan//nan	mill//2//nan//nan
1	mill//2//nan//nan	mill//2//nan//nan
2	mill//2//nan//nan	mill//2//nan//nan
3	mill//2//nan//nan	mill//2//nan//nan
4	mill//1//nan//nan	mill//1//nan//nan
5	mill//2//nan//nan	mill//2//nan//nan
6	mill//2//nan//nan	mill//2//nan//nan
7	mill//3//nan//nan	mill//3//nan//nan
8	mill//2//nan//nan	mill//2//nan//nan
9	mill//2//nan//nan	mill//2//nan//nan
10	mill//3//nan//nan	mill//3//nan//nan
11	mill//6//nan//nan	mill//6//nan//nan
12	mill//3//16//nan	mill//3//nan//nan

[190]:
```
users_receipt['new_buyer_tuple'] = np.nan

users_receipt['new_buyer_tuple'] = [check_new_buyer(users_receipt,row)
                                    for row in users_receipt.index
                                   ]
```

[191]:
```
users_receipt['new_buyer_tuple_special_group'] = np.nan

users_receipt['new_buyer_tuple_special_group'] =␣
↪[check_new_buyer_robust_special_group(users_receipt,row)
                                                    for row in users_receipt.index
                                                   ]
```

[192]:
```
users_receipt['new_buyer'] = [row[0] for row in␣
↪users_receipt['new_buyer_tuple']]
```

[193]:
```
users_receipt['new_buyer_robust_special_group'] = [row[0] for row in␣
↪users_receipt['new_buyer_tuple_special_group']]
```

[194]:
```
users_receipt['buyer'].isna().sum()
```

[194]:
```
0
```

[195]:
```
def check_new_buyer_recheck(data,row):
    user_id = data['user_id'].loc[row]
```

```
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])
    buyer_receipt_list = list(old_buyers_df['buyer_receipt'].loc[mask])
    trans_type_list = list(old_buyers_df['trans_type'].loc[mask].unique())
    new_buyer_receipt = data['new_buyer_receipt'].loc[row][0]
    buyer_receipt = data['buyer_receipt'].loc[row][0]
    trans_type = data['trans_type'].loc[row]
    ## do_id does not exist if trans_type == 'mill' and do_id_marker =='/nan//
↪nan'
    do_id_marker = data['buyer'].loc[row][-9:]
    mill_id = data['mill_id'].loc[row]
    buyer = data['buyer'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    if isnan(data['buyer'].loc[row]):
        output = np.nan
    elif trans_type== 'lr':
        if buyer!= 'lr//nan//nan//nan':
            if buyer in buyer_list:
                output = False
            else:
                output = True
        elif buyer== 'lr//nan//nan//nan':
            if 'lr' in trans_type_list:
                output = False
            else:
                output = True
    elif trans_type== 'mill':
        if buyer!= 'mill//nan//nan//nan':
            if buyer_receipt in buyer_receipt_list:
                if do_id_marker !='/nan//nan':
                    if buyer in buyer_list:
                        output=False
                    else:
                        output=True
                elif do_id_marker =='/nan//nan':
                    output=False
            else:
                output=True
        elif buyer== 'mill//nan//nan//nan':
            if 'mill' in trans_type_list:
                output = False
            else:
                output = True
```

```
    return output
```

[196]:
```
def check_new_buyer_special_group_recheck(data,row):
    user_id = data['user_id'].loc[row]
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])
    buyer_receipt_list = list(old_buyers_df['buyer_receipt'].loc[mask])
    trans_type_list = list(old_buyers_df['trans_type'].loc[mask].unique())
    new_buyer_receipt = data['new_buyer_receipt'].loc[row][0]
    buyer_receipt = data['buyer_receipt'].loc[row][0]
    trans_type = data['trans_type'].loc[row]
    ## do_id does not exist if trans_type == 'mill' and do_id_marker =='/nan//
 ↪nan'
    do_id_marker = data['buyer'].loc[row][-9:]
    mill_id = data['mill_id'].loc[row]
    buyer = data['buyer'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]

    if isnan(data['buyer'].loc[row]):
        output = np.nan
    elif trans_type== 'lr':
        if buyer!= 'lr//nan//nan//nan':
            if buyer in buyer_list:
                output = False
            else:
                output = True
        elif buyer== 'lr//nan//nan//nan':
            if 'lr' in trans_type_list:
                output = False
            else:
                output = True
    elif trans_type== 'mill':
        if buyer!= 'mill//nan//nan//nan':
            if buyer_receipt in buyer_receipt_list:
                if do_id_marker !='/nan//nan':
                    if buyer in buyer_list:
                        output=False
                    else:
                        output=True
                elif do_id_marker =='/nan//nan':
                    output=False
            else:
                output=True
        elif buyer== 'mill//nan//nan//nan':
```

```
            if 'mill' in trans_type_list:
                output = False
            else:
                output = True

    return output
```

[197]:
```
check_new_buyer_recheck(users_receipt,0)
```

[197]:
```
False
```

[198]:
```
users_receipt['new_buyer_recheck'] = [check_new_buyer_recheck(users_receipt,row)
                                      for row in users_receipt.index
                                     ]
```

[199]:
```
users_receipt['new_buyer_special_group_recheck'] =␣
↪[check_new_buyer_special_group_recheck(users_receipt,row)
                                      for row in users_receipt.index
                                     ]
```

[200]:
```
(users_receipt['new_buyer_recheck']==users_receipt['new_buyer']).describe()
```

[200]:
```
count     7933
unique       1
top       True
freq      7933
dtype: object
```

[201]:
```
(users_receipt['new_buyer_special_group_recheck']==users_receipt['new_buyer_robust_special_gro
↪describe()
```

[201]:
```
count     7933
unique       1
top       True
freq      7933
dtype: object
```

[202]:
```
def get_counterfactual_price_max(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    price = data['price_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])
```

```
    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            index = tmp_df['price_counterfactual_table'].idxmax()
            output = tmp_df['price_counterfactual_table'].loc[index]

    return output
```

[203]:
```
users_receipt['counterfactual_price_max'] = np.nan
```

[205]:
```
users_receipt['counterfactual_price_max'] = [
    get_counterfactual_price_max(users_receipt,row)
    for row in users_receipt.index
]
```

[206]:
```
def get_counterfactual_price_mean(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    price = data['price_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
```

```
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.nanmean(tmp_df['price_counterfactual_table'])

    return output
```

[207]:
```
users_receipt['counterfactual_price_mean'] = np.nan
```

[208]:
```
users_receipt['counterfactual_price_mean'] = [
    get_counterfactual_price_mean(users_receipt,row)
    for row in users_receipt.index
]
```

[209]:
```
users_receipt.shape
```

[209]:
```
(7933, 94)
```

[210]:
```
def get_counterfactual_price_special_group_robust(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    price = data['price_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.nanmean(tmp_df['price_counterfactual_table'])

    return output
```

[211]:
```
users_receipt['counterfactual_price_special_group_robust'] = np.nan
```

```
users_receipt['counterfactual_price_special_group_robust'] = [
    get_counterfactual_price_special_group_robust(users_receipt,row)
    for row in users_receipt.index
]
```

[212]:
```
def get_counterfactual_price_special_group_robust_max(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    price = data['price_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.nanmax(tmp_df['price_counterfactual_table'])

    return output
```

[213]:
```
users_receipt['counterfactual_price_special_group_robust_max'] = np.nan

users_receipt['counterfactual_price_special_group_robust_max'] = [
    get_counterfactual_price_special_group_robust_max(users_receipt,row)
    for row in users_receipt.index
]
```

[214]:
```
def get_counterfactual_post_grading_price_max(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    post_grading_price = data['post_grading_price'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
```

```
            )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = post_grading_price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','post_grading_price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            index = tmp_df['post_grading_price_counterfactual_table'].idxmax()
            output = tmp_df['post_grading_price_counterfactual_table'].
↪loc[index]

    return output
```

[215]:
```
users_receipt['counterfactual_post_grading_price_max'] = np.nan
```

[216]:
```
users_receipt['counterfactual_post_grading_price_max'] = [
    get_counterfactual_post_grading_price_max(users_receipt,row)
    for row in users_receipt.index
]
```

[217]:
```
counterfactual_price.columns
```

[217]:
```
Index(['date', 'buyer', 'buyer_receipt', 'grade_cleaned', 'price_cleaned',
       'max_price_cleaned', 'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa'],
      dtype='object')
```

[218]:
```
def get_counterfactual_post_grading_price_mean(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    post_grading_price = data['post_grading_price'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
```

```
            )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = post_grading_price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','post_grading_price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.
↪nanmean(tmp_df['post_grading_price_counterfactual_table'])

    return output
```

[219]:
```
users_receipt['counterfactual_post_grading_price_mean'] = np.nan
```

[220]:
```
users_receipt['counterfactual_post_grading_price_mean'] = [
    get_counterfactual_post_grading_price_mean(users_receipt,row)
    for row in users_receipt.index
]
```

[221]:
```
def get_counterfactual_post_grading_price_special_group_robust(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    post_grading_price = data['post_grading_price'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = post_grading_price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
```

```
        cols = ['date','buyer','post_grading_price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.
↪nanmean(tmp_df['post_grading_price_counterfactual_table'])

    return output
```

[222]:
```
users_receipt['counterfactual_post_grading_price_special_group_robust'] = np.nan

users_receipt['counterfactual_post_grading_price_special_group_robust'] = [
    ␣
↪get_counterfactual_post_grading_price_special_group_robust(users_receipt,row)
    for row in users_receipt.index
]
```

[223]:
```
def get_counterfactual_post_grading_price_special_group_robust_max(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    post_grading_price = data['post_grading_price'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) #&
            #(old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = post_grading_price
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','post_grading_price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.
↪nanmax(tmp_df['post_grading_price_counterfactual_table'])

    return output
```

[224]:
```
users_receipt['counterfactual_post_grading_price_special_group_robust_max'] =␣
↪np.nan

users_receipt['counterfactual_post_grading_price_special_group_robust_max'] = [
   ␣
↪get_counterfactual_post_grading_price_special_group_robust_max(users_receipt,row)
    for row in users_receipt.index
]
```

[225]:
```
def get_counterfactual_grade_max(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    grade = data['grade_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = grade
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols =␣
↪['date','buyer','post_grading_price_counterfactual_table','grade_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            index = tmp_df['post_grading_price_counterfactual_table'].idxmax()
            output = tmp_df['grade_counterfactual_table'].loc[index]

    return output
```

[226]:
```
users_receipt['counterfactual_grade_max'] = np.nan
```

[227]:
```
users_receipt['counterfactual_grade_max'] = [
    get_counterfactual_grade_max(users_receipt,row)
    for row in users_receipt.index
]
```

[228]:
```
def get_counterfactual_grade_mean(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    grade = data['grade_imputed_final'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
    elif data['new_buyer'].loc[row] == False:
        output = grade
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','grade_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            #index = tmp_df['post_grading_price'].idxmax()
            output = np.nanmean(tmp_df['grade_counterfactual_table'])

    return output
```

[229]:
```
users_receipt['counterfactual_grade_mean'] = np.nan
```

[230]:
```
users_receipt['counterfactual_grade_mean'] = [
    get_counterfactual_grade_mean(users_receipt,row)
    for row in users_receipt.index
]
```

[231]:
```
def get_counterfactual_buyer(data,row):
    user_id = data['user_id'].loc[row]
    date = data['date'].loc[row]
    buyer = data['buyer'].loc[row]
    ## only old buyers from that are from in-app survey and 2018 survey
    mask = ((old_buyers_df['user_id']==user_id) &
            (old_buyers_df['source']!= 'special price membership')
           )
    buyer_list = list(old_buyers_df['buyer'].loc[mask])

    if isnan(data['new_buyer'].loc[row]):
        output = np.nan
```

```
    elif data['new_buyer'].loc[row] == False:
        output = buyer
    else:
        mask_2 = ((counterfactual_price['date']==date) &
                  (counterfactual_price['buyer'].isin(buyer_list))
                 )
        cols = ['date','buyer','post_grading_price_counterfactual_table']
        tmp_df = counterfactual_price[cols][mask_2].dropna()
        if tmp_df.empty:
            output=np.nan
        else:
            index = tmp_df['post_grading_price_counterfactual_table'].idxmax()
            output = tmp_df['buyer'].loc[index]

    return output
```

[232]:
```
users_receipt['counterfactual_buyer'] = np.nan
```

[233]:
```
users_receipt['counterfactual_buyer'] = [
    get_counterfactual_buyer(users_receipt,row)
    for row in users_receipt.index
]
```

[234]:
```
print(users_receipt.shape)

cols = ['counterfactual_buyer','counterfactual_buyer_kecamatan_desa']

users_receipt = users_receipt.merge(buyers[cols],
                                    on = 'counterfactual_buyer',
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 103)
(7933, 104)
```

[235]:
```
users_receipt['counterfactual_buyer_in_the_same_village'] =␣
↪(users_receipt['kecamatan_desa'] ==
                                                          ␣
↪users_receipt['counterfactual_buyer_kecamatan_desa']
                                                          )
```

[236]:
```
users_receipt['post_grading_price_difference'] =␣
↪(users_receipt['post_grading_price'] -
                                                  ␣
↪users_receipt['counterfactual_post_grading_price_mean']
```

```
                                                    )
```

[237]:
```
users_receipt['post_grading_price_difference_%'] =␣
↪(users_receipt['post_grading_price_difference']/
                                                   ␣
↪users_receipt['counterfactual_post_grading_price_mean']
                                                    )*100
```

[238]:
```
(users_receipt['price_imputed_final']-users_receipt['counterfactual_price_max']).
↪describe()
```

[238]:
```
count    5673.000000
mean        0.063635
std        39.854215
min      -560.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       265.000000
dtype: float64
```

[239]:
```
mask =␣
↪(users_receipt['price_imputed_final']-users_receipt['counterfactual_price_max']␣
↪<-300)

users_receipt[['user_id','price_imputed_final','counterfactual_price_max','price_source','buye
```

[239]:
```
      user_id  price_imputed_final  counterfactual_price_max      price_source  \
5150    1459               1500.0                    1885.0  price fruit sale
5169    1459               1550.0                    1885.0  price fruit sale
7590     608                950.0                    1505.0  Transactions_sql
7591     608                950.0                    1510.0  Transactions_sql

                    buyer        date
5150    mill//2//15//nan  2020-10-30
5169   lr//nan//nan//107  2020-10-30
7590  mill//47//193//nan  2020-03-11
7591  mill//47//193//nan  2020-03-12
```

[240]:
```
mask =␣
↪(users_receipt['price_imputed_final']-users_receipt['counterfactual_price_mean']␣
↪<-300)

users_receipt[['user_id','price_imputed_final','counterfactual_price_max','price_source','buye
```

[240]:
```
      user_id  price_imputed_final  counterfactual_price_max      price_source  \
5150    1459               1500.0                    1885.0  price fruit sale
7590     608                950.0                    1505.0  Transactions_sql
7591     608                950.0                    1510.0  Transactions_sql

                    buyer        date
5150    mill//2//15//nan  2020-10-30
7590  mill//47//193//nan  2020-03-11
7591  mill//47//193//nan  2020-03-12
```

[241]:
```
mask = (users_receipt['post_grading_price_difference'] <-300)

users_receipt[['user_id','price_imputed_final','counterfactual_price_mean','price_source','buy
```

[241]:
```
      user_id  price_imputed_final  counterfactual_price_mean  \
5150    1459               1500.0                     1825.0

          price_source             buyer        date  special_price
5150  price fruit sale  mill//2//15//nan  2020-10-30            NaN
```

[242]:
```
mask = ((counterfactual_price['buyer']=='lr//nan//nan//54') &
        (counterfactual_price['date'] >= pd.to_datetime('2020-03-21')) &
        (counterfactual_price['date'] <= pd.to_datetime('2020-03-31'))
       )

counterfactual_price[mask]
```

[242]:
```
             date             buyer     buyer_receipt  grade_cleaned  \
28581  2020-03-21  lr//nan//nan//54  lr//nan//nan//54            NaN
28582  2020-03-22  lr//nan//nan//54  lr//nan//nan//54            NaN
28583  2020-03-23  lr//nan//nan//54  lr//nan//nan//54            NaN
28584  2020-03-24  lr//nan//nan//54  lr//nan//nan//54            NaN
28585  2020-03-25  lr//nan//nan//54  lr//nan//nan//54            NaN
28586  2020-03-26  lr//nan//nan//54  lr//nan//nan//54            NaN
28587  2020-03-27  lr//nan//nan//54  lr//nan//nan//54            NaN
28588  2020-03-28  lr//nan//nan//54  lr//nan//nan//54           3.50
28589  2020-03-29  lr//nan//nan//54  lr//nan//nan//54            NaN
28590  2020-03-30  lr//nan//nan//54  lr//nan//nan//54           3.53
28591  2020-03-31  lr//nan//nan//54  lr//nan//nan//54           3.49

       price_cleaned  max_price_cleaned  min_grade_cleaned  \
28581            NaN                NaN                NaN
28582            NaN                NaN                NaN
28583            NaN                NaN                NaN
28584            NaN                NaN                NaN
28585            NaN                NaN                NaN
28586            NaN                NaN                NaN
```

```
28587           NaN               NaN               NaN
28588           NaN               NaN              3.50
28589           NaN               NaN               NaN
28590        1590.0            1590.0              3.53
28591        1600.0            1600.0              3.39

       grade_cleaned_buyer_receipt  price_cleaned_buyer_receipt  \
28581                          NaN                          NaN
28582                          NaN                          NaN
28583                          NaN                          NaN
28584                          NaN                          NaN
28585                          NaN                          NaN
28586                          NaN                          NaN
28587                          NaN                          NaN
28588                         3.50                          NaN
28589                          NaN                          NaN
28590                         3.53                       1590.0
28591                         3.49                       1600.0

       max_price_cleaned_buyer_receipt  min_grade_cleaned_buyer_receipt  \
28581                              NaN                              NaN
28582                              NaN                              NaN
28583                              NaN                              NaN
28584                              NaN                              NaN
28585                              NaN                              NaN
28586                              NaN                              NaN
28587                              NaN                              NaN
28588                              NaN                             3.50
28589                              NaN                              NaN
28590                           1590.0                             3.53
28591                           1600.0                             3.39

       price_counterfactual_table  grade_counterfactual_table  \
28581                         NaN                         NaN
28582                         NaN                         NaN
28583                         NaN                         NaN
28584                         NaN                         NaN
28585                         NaN                         NaN
28586                         NaN                         NaN
28587                         NaN                         NaN
28588                         NaN                        3.50
28589                         NaN                         NaN
28590                      1590.0                        3.53
28591                      1600.0                        3.49

       post_grading_price_counterfactual_table  \
28581                                      NaN
```

```
28582                                    NaN
28583                                    NaN
28584                                    NaN
28585                                    NaN
28586                                    NaN
28587                                    NaN
28588                                    NaN
28589                                    NaN
28590                                1533.873
28591                                1544.160

       max_price_counterfactual_table  min_grade_counterfactual_table  \
28581                             NaN                             NaN
28582                             NaN                             NaN
28583                             NaN                             NaN
28584                             NaN                             NaN
28585                             NaN                             NaN
28586                             NaN                             NaN
28587                             NaN                             NaN
28588                             NaN                            3.50
28589                             NaN                             NaN
28590                          1590.0                            3.53
28591                          1600.0                            3.39

       max_post_grading_price_counterfactual_table     buyer_kecamatan_desa
28581                                          NaN  BATANG GANSAL__SEBERIDA
28582                                          NaN  BATANG GANSAL__SEBERIDA
28583                                          NaN  BATANG GANSAL__SEBERIDA
28584                                          NaN  BATANG GANSAL__SEBERIDA
28585                                          NaN  BATANG GANSAL__SEBERIDA
28586                                          NaN  BATANG GANSAL__SEBERIDA
28587                                          NaN  BATANG GANSAL__SEBERIDA
28588                                          NaN  BATANG GANSAL__SEBERIDA
28589                                          NaN  BATANG GANSAL__SEBERIDA
28590                                     1533.873  BATANG GANSAL__SEBERIDA
28591                                     1545.760  BATANG GANSAL__SEBERIDA
```

[243]:
```
mask = ((counterfactual_price['buyer']=='mill//2//26//nan') &
        (counterfactual_price['date'] >= pd.to_datetime('2020-10-25')) &
        (counterfactual_price['date'] <= pd.to_datetime('2020-11-05'))
       )

counterfactual_price[mask]
```

[243]:
```
            date             buyer      buyer_receipt  grade_cleaned  \
73767  2020-10-25  mill//2//26//nan  mill//2//nan//nan            NaN
73768  2020-10-26  mill//2//26//nan  mill//2//nan//nan            NaN
```

```
73769  2020-10-27  mill//2//26//nan  mill//2//nan//nan            NaN
73770  2020-10-28  mill//2//26//nan  mill//2//nan//nan            NaN
73771  2020-10-29  mill//2//26//nan  mill//2//nan//nan            NaN
73772  2020-10-30  mill//2//26//nan  mill//2//nan//nan            NaN
73773  2020-10-31  mill//2//26//nan  mill//2//nan//nan            NaN
73774  2020-11-01  mill//2//26//nan  mill//2//nan//nan            NaN
73775  2020-11-02  mill//2//26//nan  mill//2//nan//nan            NaN
73776  2020-11-03  mill//2//26//nan  mill//2//nan//nan            NaN
73777  2020-11-04  mill//2//26//nan  mill//2//nan//nan            4.5
73778  2020-11-05  mill//2//26//nan  mill//2//nan//nan            NaN

       price_cleaned  max_price_cleaned  min_grade_cleaned  \
73767            NaN                NaN                NaN
73768         1865.0             1865.0                NaN
73769         1885.0             1885.0                NaN
73770         1885.0             1885.0                NaN
73771         1865.0             1865.0                NaN
73772            NaN                NaN                NaN
73773            NaN                NaN                NaN
73774         1865.0             1865.0                NaN
73775         1865.0             1865.0                NaN
73776         1650.0             1650.0                NaN
73777         1885.0             1885.0                4.5
73778            NaN                NaN                NaN

       grade_cleaned_buyer_receipt  price_cleaned_buyer_receipt  \
73767                          NaN                          NaN
73768                        4.495                       1865.0
73769                        4.750                       1885.0
73770                        4.500                       1885.0
73771                        5.130                       1865.0
73772                        5.000                       1885.0
73773                        4.990                       1880.0
73774                        4.755                       1865.0
73775                        4.750                       1880.0
73776                        4.990                       1865.0
73777                        4.500                       1885.0
73778                        4.500                       1892.5

       max_price_cleaned_buyer_receipt  min_grade_cleaned_buyer_receipt  \
73767                              NaN                              NaN
73768                           1880.0                             4.00
73769                           1900.0                             4.70
73770                           1900.0                             4.49
73771                           1880.0                             4.49
73772                           1900.0                             4.76
73773                           1880.0                             4.75
```

```
73774                          1865.0                              4.75
73775                          1990.0                              4.00
73776                          1880.0                              4.50
73777                          1885.0                              4.50
73778                          1915.0                              4.50

       price_counterfactual_table  grade_counterfactual_table  \
73767                         NaN                         NaN
73768                      1865.0                       4.495
73769                      1885.0                       4.750
73770                      1885.0                       4.500
73771                      1865.0                       5.130
73772                      1885.0                       5.000
73773                      1880.0                       4.990
73774                      1865.0                       4.755
73775                      1865.0                       4.750
73776                      1650.0                       4.990
73777                      1885.0                       4.500
73778                      1892.5                       4.500

       post_grading_price_counterfactual_table  \
73767                                      NaN
73768                                1781.16825
73769                                1795.46250
73770                                1800.17500
73771                                1769.32550
73772                                1790.75000
73773                                1786.18800
73774                                1776.31925
73775                                1776.41250
73776                                1567.66500
73777                                1800.17500
73778                                1807.33750

       max_price_counterfactual_table  min_grade_counterfactual_table  \
73767                             NaN                             NaN
73768                          1865.0                            4.00
73769                          1885.0                            4.70
73770                          1885.0                            4.49
73771                          1865.0                            4.49
73772                          1900.0                            4.76
73773                          1880.0                            4.75
73774                          1865.0                            4.75
73775                          1865.0                            4.00
73776                          1650.0                            4.50
73777                          1885.0                            4.50
73778                          1915.0                            4.50
```

```
       max_post_grading_price_counterfactual_table   buyer_kecamatan_desa
73767                                          NaN  BATANG GANSAL__RINGIN
73768                                    1790.4000  BATANG GANSAL__RINGIN
73769                                    1796.4050  BATANG GANSAL__RINGIN
73770                                    1800.3635  BATANG GANSAL__RINGIN
73771                                    1781.2615  BATANG GANSAL__RINGIN
73772                                    1809.5600  BATANG GANSAL__RINGIN
73773                                    1790.7000  BATANG GANSAL__RINGIN
73774                                    1776.4125  BATANG GANSAL__RINGIN
73775                                    1790.4000  BATANG GANSAL__RINGIN
73776                                    1575.7500  BATANG GANSAL__RINGIN
73777                                    1800.1750  BATANG GANSAL__RINGIN
73778                                    1828.8250  BATANG GANSAL__RINGIN
```

[244]:
```
mask = (users_receipt['price_imputed_final'] ==1000)

users_receipt[['user_id','price_imputed_final','counterfactual_price_mean','price_source','buy
```

[244]:
```
    user_id  price_imputed_final  counterfactual_price_mean      price_source  \
82      124               1000.0                        NaN  Transactions_sql
436      59               1000.0                        NaN  Transactions_sql
438      59               1000.0                        NaN  Transactions_sql
817     124               1000.0                        NaN  Transactions_sql

                buyer        date
82   mill//1//24//nan  2019-01-23
436   lr//nan//nan//2  2019-06-18
438   lr//nan//nan//2  2019-06-19
817  mill//8//27//nan  2019-09-20
```

[245]:
```
users_receipt['price_source']
```

[245]:
```
0                Manual Imputation
1       Counterfactual price table
2                Manual Imputation
3                Manual Imputation
4                Manual Imputation
                   …
7928    Counterfactual price table
7929                           NaN
7930    Counterfactual price table
7931           special_price_daily
7932    Counterfactual price table
Name: price_source, Length: 7933, dtype: object
```

[246]:
```
mask = (users_receipt['post_grading_price_difference'] <-300)

print(users_receipt.shape)
print(users_receipt[mask].shape)
users_receipt['post_grading_price_difference'][mask]

#users_receipt = users_receipt[mask]
```

```
(7933, 107)
(1, 107)
```

[246]:
```
5150   -311.51875
Name: post_grading_price_difference, dtype: float64
```

[247]:
```
users_receipt['post_grading_price_difference'].describe()
```

[247]:
```
count    5312.000000
mean        7.008432
std        36.337472
min      -311.518750
25%         0.000000
50%         0.000000
75%         0.000000
max       305.401322
Name: post_grading_price_difference, dtype: float64
```

3.1.12 Calculate best price and best price viewed

[248]:
```
best_price_daily_users_receipt = pd.pivot_table(users_receipt,
                                                index='date',
                                                values =␣
↪['price_imputed_final','post_grading_price'],
                                                aggfunc= np.max
                                                )

best_price_daily_users_receipt.rename(columns= {'price_imputed_final':
↪'best_price_users_receipt',
                                                'post_grading_price':
↪'best_post_grading_price_users_receipt'},
                                      inplace = True
                                     )

best_price_daily_users_receipt.reset_index(inplace=True)

best_price_daily_users_receipt['date'] = pd.
↪to_datetime(best_price_daily_users_receipt['date']).dt.date
```

```
best_price_daily_users_receipt.tail()
```

[248]:
```
           date  best_post_grading_price_users_receipt  \
770  2021-02-10                                    NaN
771  2021-02-13                               1866.750
772  2021-02-14                               1722.240
773  2021-02-15                               1896.395
774  2021-02-19                               1909.007

     best_price_users_receipt
770                    1850.0
771                    1940.0
772                    1800.0
773                    1975.0
774                    1990.0
```

[249]:
```
counterfactual_price.head()
```

[249]:
```
         date             buyer buyer_receipt  grade_cleaned  price_cleaned  \
0  2018-12-01  lr//nan//nan//11           NaN            NaN            NaN
1  2018-12-02  lr//nan//nan//11           NaN            NaN            NaN
2  2018-12-03  lr//nan//nan//11           NaN            NaN            NaN
3  2018-12-04  lr//nan//nan//11           NaN            NaN            NaN
4  2018-12-05  lr//nan//nan//11           NaN            NaN            NaN

   max_price_cleaned  min_grade_cleaned  grade_cleaned_buyer_receipt  \
0                NaN                NaN                          NaN
1                NaN                NaN                          NaN
2                NaN                NaN                          NaN
3                NaN                NaN                          NaN
4                NaN                NaN                          NaN

   price_cleaned_buyer_receipt  max_price_cleaned_buyer_receipt  \
0                          NaN                              NaN
1                          NaN                              NaN
2                          NaN                              NaN
3                          NaN                              NaN
4                          NaN                              NaN

   min_grade_cleaned_buyer_receipt  price_counterfactual_table  \
0                              NaN                         NaN
1                              NaN                         NaN
2                              NaN                         NaN
3                              NaN                         NaN
4                              NaN                         NaN

   grade_counterfactual_table  post_grading_price_counterfactual_table  \
```

```
0                           NaN                                    NaN
1                           NaN                                    NaN
2                           NaN                                    NaN
3                           NaN                                    NaN
4                           NaN                                    NaN

   max_price_counterfactual_table  min_grade_counterfactual_table  \
0                             NaN                             NaN
1                             NaN                             NaN
2                             NaN                             NaN
3                             NaN                             NaN
4                             NaN                             NaN

   max_post_grading_price_counterfactual_table buyer_kecamatan_desa
0                                          NaN                  NaN
1                                          NaN                  NaN
2                                          NaN                  NaN
3                                          NaN                  NaN
4                                          NaN                  NaN
```

[250]:
```
'''
cols = ['max_post_grading_price','nanmax_post_grading_price_users_receipt']

counterfactual_price['max_post_grading_price']=counterfactual_price[cols].
 ↪max(axis=1)
'''
```

[250]:
```
"\ncols = ['max_post_grading_price','nanmax_post_grading_price_users_receipt']\n
\ncounterfactual_price['max_post_grading_price']=counterfactual_price[cols].max(
axis=1)\n"
```

[251]:
```
'''
cols = ['max_price_imputed_final','nanmax_price_imputed_final_users_receipt']

counterfactual_price['max_price_imputed_final']=counterfactual_price[cols].
 ↪max(axis=1)
'''
```

[251]:
```
"\ncols = ['max_price_imputed_final','nanmax_price_imputed_final_users_receipt']
\n\ncounterfactual_price['max_price_imputed_final']=counterfactual_price[cols].m
ax(axis=1)\n"
```

[252]:
```
best_price_daily = pd.pivot_table(counterfactual_price,
                                  index='date',
                                  values = ['max_price_counterfactual_table',
                                                                  ␣
 ↪'max_post_grading_price_counterfactual_table'],
```

```
                                    aggfunc= np.nanmax
                                   )

#best_price_daily.rename(columns= {'max_price_counterfactual_table':
 ↪'best_price',
#                                  ␣
 ↪'max_post_grading_price_counterfactual_table':'best_post_grading_price'},
#                          inplace = True
#                         )

best_price_daily.reset_index(inplace=True)

best_price_daily['date'] = pd.to_datetime(best_price_daily['date']).dt.date

best_price_daily.tail()
```

[252]:

	date	max_post_grading_price_counterfactual_table	\
797	2021-02-06	1891.200	
798	2021-02-07	1988.500	
799	2021-02-08	1939.650	
800	2021-02-09	1953.035	
801	2021-02-10	1843.008	

	max_price_counterfactual_table
797	2030.0
798	2050.0
799	2060.0
800	2080.0
801	1955.0

[253]:
```
print(best_price_daily.shape)

best_price_daily = best_price_daily.merge(best_price_daily_users_receipt,
                                          how ='left',
                                          on = 'date'
                                         )

print(best_price_daily.shape)
```

```
(802, 3)
(802, 5)
```

[254]:
```
best_price_daily.head()
```

[254]:

	date	max_post_grading_price_counterfactual_table	\
0	2018-12-02	NaN	
1	2018-12-03	NaN	

```
2  2018-12-04                                        NaN
3  2018-12-05                                        NaN
4  2018-12-06                                    969.825

   max_price_counterfactual_table  best_post_grading_price_users_receipt  \
0                           975.0                                    NaN
1                          1065.0                                    NaN
2                          1065.0                                    NaN
3                          1065.0                                    NaN
4                          1065.0                                  893.0

   best_price_users_receipt
0                       NaN
1                       NaN
2                       NaN
3                     940.0
4                     940.0
```

[255]:
```
cols = ␣
 ↪['max_post_grading_price_counterfactual_table','best_post_grading_price_users_receipt']

best_price_daily['best_post_grading_price'] = best_price_daily[cols].max(axis=1)
```

[256]:
```
cols = ['max_price_counterfactual_table','best_price_users_receipt']

best_price_daily['best_price'] = best_price_daily[cols].max(axis=1)
```

[257]:
```
#best_price_daily['best_post_grading_price'] = ␣
 ↪best_price_daily['best_post_grading_price_counterfactual_price']

#best_price_daily['best_post_grading_price'].
 ↪fillna(best_price_daily['best_post_grading_price_users_receipt'],␣
 ↪inplace=True)

#best_price_daily['best_post_grading_price'].isna().sum()
```

[258]:
```
best_price_daily.head()
```

[258]:
```
         date  max_post_grading_price_counterfactual_table  \
0  2018-12-02                                        NaN
1  2018-12-03                                        NaN
2  2018-12-04                                        NaN
3  2018-12-05                                        NaN
4  2018-12-06                                    969.825

   max_price_counterfactual_table  best_post_grading_price_users_receipt  \
0                           975.0                                    NaN
```

```
1                          1065.0                                  NaN
2                          1065.0                                  NaN
3                          1065.0                                  NaN
4                          1065.0                                893.0

   best_price_users_receipt  best_post_grading_price  best_price
0                       NaN                      NaN       975.0
1                       NaN                      NaN      1065.0
2                       NaN                      NaN      1065.0
3                     940.0                      NaN      1065.0
4                     940.0                  969.825      1065.0
```

[259]:
```
best_price_daily['best_price'].dropna().shape
```

[259]:
```
(802,)
```

[260]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(best_price_daily,
                                    on='date',
                                    how='left'
                                   )

print(users_receipt.shape)
```

```
(7933, 107)
(7933, 113)
```

[261]:
```
do_ramp_visit.head()
```

[261]:
```
   id user_id  do_id                 date_time        date    month  week  \
0   3     124    NaN 2019-05-09 03:28:18.000000  2019-05-09  2019-05    19
1   5     201    NaN 2019-05-09 06:27:29.000000  2019-05-09  2019-05    19
2   6     201    NaN 2019-05-09 06:27:33.000000  2019-05-09  2019-05    19
3   7     201    NaN 2019-05-09 06:27:35.000001  2019-05-09  2019-05    19
4   8     201    NaN 2019-05-09 06:27:39.000000  2019-05-09  2019-05    19

   biweek  triweek  quadweek  …  village_cleaned      kecamatan  \
0      10        7         5  …         Seberida  Batang Gansal
1      10        7         5  …         Seberida  Batang Gansal
2      10        7         5  …         Seberida  Batang Gansal
3      10        7         5  …         Seberida  Batang Gansal
4      10        7         5  …         Seberida  Batang Gansal

            kecamatan_desa membership_length new_buyer  \
0  BATANG GANSAL__SEBERIDA               107       1.0
1  BATANG GANSAL__SEBERIDA                28       1.0
```

```
2  BATANG GANSAL__SEBERIDA                28       1.0
3  BATANG GANSAL__SEBERIDA                28       1.0
4  BATANG GANSAL__SEBERIDA                28       1.0

          buyer_kecamatan_desa kecamatan_desa_connected same_village  \
0                          NaN                      NaN          NaN
1    SEBERIDA__PANGKALAN KASAI                      0.0          0.0
2   BATANG CENAKU__BUKIT LIPAI                      0.0          0.0
3      BATANG CENAKU__AUR CINA                      0.0          0.0
4  RENGAT BARAT__PEMATANG REBA                      0.0          0.0

       user_kecamatan_desa  shortest_path_length
0                      NaN                   NaN
1  BATANG GANSAL__SEBERIDA                   3.0
2  BATANG GANSAL__SEBERIDA                   3.0
3  BATANG GANSAL__SEBERIDA                   2.0
4  BATANG GANSAL__SEBERIDA                   4.0

[5 rows x 51 columns]
```

[262]:
```
do_ramp_visit.columns
```

[262]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt', 'grade_cleaned', 'price_cleaned', 'max_price_cleaned',
       'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa_x',
       'join_date', 'village_cleaned', 'kecamatan', 'kecamatan_desa',
       'membership_length', 'new_buyer', 'buyer_kecamatan_desa',
       'kecamatan_desa_connected', 'same_village', 'user_kecamatan_desa',
       'shortest_path_length'],
      dtype='object')
```

[263]:
```
'''
print(do_ramp_visit.shape)

do_ramp_visit = do_ramp_visit.merge(users_receipt_price,
                                    on = ['buyer','date'],
                                    how = 'left'
                                    )
```

```
print(do_ramp_visit.shape)
'''
```

[263]:
```
"\nprint(do_ramp_visit.shape)\n\ndo_ramp_visit =
do_ramp_visit.merge(users_receipt_price, \n
on = ['buyer','date'],\n                                    how = 'left'\n
)\n\nprint(do_ramp_visit.shape)\n"
```

[264]:
```
'''
do_ramp_visit['post_grading_price'].
↪fillna(do_ramp_visit['nanmedian_post_grading_price_users_receipt'],␣
↪inplace=True)

do_ramp_visit['price_imputed_final'].
↪fillna(do_ramp_visit['nanmedian_price_imputed_final_users_receipt'],␣
↪inplace=True)
'''
```

[264]:
```
"\ndo_ramp_visit['post_grading_price'].fillna(do_ramp_visit['nanmedian_post_grad
ing_price_users_receipt'], inplace=True)\n\ndo_ramp_visit['price_imputed_final']
.fillna(do_ramp_visit['nanmedian_price_imputed_final_users_receipt'],
inplace=True)\n"
```

[265]:
```
do_ramp_visit.columns
```

[265]:
```
Index(['id', 'user_id', 'do_id', 'date_time', 'date', 'month', 'week',
       'biweek', 'triweek', 'quadweek', 'year', 'year_week', 'year_biweek',
       'year_triweek', 'year_quadweek', 'name', 'phone', 'role', 'mill_id',
       'price_sharing', 'Sharing or Viewing', 'trans_type', 'lr_id', 'buyer',
       'buyer_receipt', 'grade_cleaned', 'price_cleaned', 'max_price_cleaned',
       'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa_x',
       'join_date', 'village_cleaned', 'kecamatan', 'kecamatan_desa',
       'membership_length', 'new_buyer', 'buyer_kecamatan_desa',
       'kecamatan_desa_connected', 'same_village', 'user_kecamatan_desa',
       'shortest_path_length'],
      dtype='object')
```

[266]:
```
best_price_viewed = pd.pivot_table(do_ramp_visit,
                                   index = ['date','user_id'],
                                   values = ['max_price_counterfactual_table',
                                             ␣
↪'max_post_grading_price_counterfactual_table'],
```

```
                                    aggfunc= np.nanmax
                                   )

best_price_viewed.rename(columns = {'max_price_counterfactual_table':
↪'best_price_viewed',
                                    ␣
↪'max_post_grading_price_counterfactual_table':
↪'best_post_grading_price_viewed'},
                         inplace = True
                        )

best_price_viewed.reset_index(inplace=True)

best_price_viewed['date'] = pd.to_datetime(best_price_viewed['date']).dt.date

best_price_viewed.head()
```

[266]:

	date	user_id	best_post_grading_price_viewed	best_price_viewed
0	2019-05-09	124	NaN	1205.0
1	2019-05-09	201	1195.74	1300.0
2	2019-05-09	214	1195.74	1300.0
3	2019-05-09	23	1189.16	1240.0
4	2019-05-09	240	1195.74	1300.0

[267]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(best_price_viewed,
                                    on=['date','user_id'],
                                    how='left'
                                   )

print(users_receipt.shape)
```

```
(7933, 113)
(7933, 115)
```

[268]:
```
'''best_grade_daily = pd.pivot_table(counterfactual_price,
                                    index='date',
                                    values = ['grade_imputed_final'],
                                    aggfunc= np.nanmin
                                   )

best_grade_daily.rename(columns= {'grade_counterfactual_table':'best_grade',
                                    #'post_grading_price':
↪'best_post_grading_price'
                                   },
                        inplace = True
```

```
                        )

best_grade_daily.reset_index(inplace=True)

best_grade_daily['date'] = pd.to_datetime(best_grade_daily['date']).dt.date

best_grade_daily.tail()
'''
```

[268]:
```
"best_grade_daily = pd.pivot_table(counterfactual_price, \n
index='date',\n                                        values =
['grade_imputed_final'],\n                                          aggfunc= np.nanmin\n
)\n\nbest_grade_daily.rename(columns=
{'grade_counterfactual_table':'best_grade',\n
#'post_grading_price':'best_post_grading_price'\n
},\n                          inplace = True\n
)\n\nbest_grade_daily.reset_index(inplace=True)\n\nbest_grade_daily['date'] =
pd.to_datetime(best_grade_daily['date']).dt.date\n\nbest_grade_daily.tail()\n"
```

[269]:
```
'''
print(users_receipt.shape)

users_receipt = users_receipt.merge(best_grade_daily,
                                                  on='date',
                                                  how='left'
                                                 )


print(users_receipt.shape)
'''
```

[269]:
```
"\nprint(users_receipt.shape)\n\nusers_receipt =
users_receipt.merge(best_grade_daily,\n
on='date',\n                                                    how='left'\n
)\n\nprint(users_receipt.shape)\n"
```

[270]:
```
'''
best_grade_viewed = pd.pivot_table(do_ramp_visit,
                                   index = ['date','user_id'],
                                   values = ['grade_imputed_final'],
                                   aggfunc= np.nanmin
                                  )

best_grade_viewed.rename(columns = {'grade_imputed_final':'best_grade_viewed',
                                    #'post_grading_price':
 ↪'best_post_grading_price_viewed'
                                   },
                         inplace = True
```

```
                        )

best_grade_viewed.reset_index(inplace=True)

best_grade_viewed['date'] = pd.to_datetime(best_grade_viewed['date']).dt.date

best_grade_viewed.head()
'''
```

[270]:
```
"\nbest_grade_viewed = pd.pivot_table(do_ramp_visit,\n
index = ['date','user_id'],\n                                        values =
['grade_imputed_final'],\n                                        aggfunc=
np.nanmin\n
)\n\nbest_grade_viewed.rename(columns =
{'grade_imputed_final':'best_grade_viewed',\n
#'post_grading_price':'best_post_grading_price_viewed'\n
},\n                           inplace = True\n
)\n\nbest_grade_viewed.reset_index(inplace=True)\n\nbest_grade_viewed['date'] =
pd.to_datetime(best_grade_viewed['date']).dt.date\n\nbest_grade_viewed.head()\n"
```

[271]:
```
'''
print(users_receipt.shape)

users_receipt = users_receipt.merge(best_grade_viewed,
                                                    on=['date','user_id'],
                                                    how='left'
                                                   )

print(users_receipt.shape)
'''
```

[271]:
```
"\nprint(users_receipt.shape)\n\nusers_receipt =
users_receipt.merge(best_grade_viewed,\n
on=['date','user_id'],\n
how='left'\n
)\n\nprint(users_receipt.shape)\n"
```

[272]:
```
users_receipt['best_price_viewed'] =␣
↪users_receipt[['best_price_viewed','price_imputed_final']].max(axis=1)
```

[273]:
```
users_receipt['best_post_grading_price_viewed'] =␣
↪users_receipt[['best_post_grading_price_viewed','post_grading_price']].
↪max(axis=1)
```

[274]:
```
mask = (users_receipt['best_post_grading_price_viewed'] -␣
↪users_receipt['post_grading_price']) <0
```

```
users_receipt[['best_post_grading_price_viewed','post_grading_price']][mask]
```

[274]:
```
Empty DataFrame
Columns: [best_post_grading_price_viewed, post_grading_price]
Index: []
```

[275]:
```
(users_receipt['best_post_grading_price_viewed'] -␣
↪users_receipt['post_grading_price']).describe()
```

[275]:
```
count    7481.000000
mean       29.069004
std        51.726080
min         0.000000
25%         0.000000
50%         3.934500
75%        33.020500
max       458.712000
dtype: float64
```

[276]:
```
(users_receipt['best_price'] - users_receipt['price_imputed_final']).describe()
```

[276]:
```
count    7739.000000
mean      110.426541
std        75.666508
min         0.000000
25%        50.000000
50%       105.000000
75%       160.000000
max       650.000000
dtype: float64
```

3.1.13 Calculate distance

[277]:
```
def get_distance(data,row):
    user_id = data['user_id'].loc[row]
    buyer = data['buyer'].loc[row]
    buyer_receipt = data['buyer_receipt'].loc[row]
    if ((user_id in user_buyer_distance.index) &
        (buyer in user_buyer_distance.columns)):
        output = user_buyer_distance[buyer].loc[user_id]
    elif ((user_id in user_buyer_receipt_distance.index) &
          (buyer_receipt in user_buyer_receipt_distance.columns)):
        output = user_buyer_receipt_distance[buyer_receipt].loc[user_id]
    else :
        output = np.nan
    return output
```

[278]:
```
last_most_transaction_do_ramp['distance'] =␣
↪[get_distance(last_most_transaction_do_ramp,row)
                                              for row in␣
↪last_most_transaction_do_ramp.index
                                             ]
```

[279]:
```
users_receipt['distance'] = [get_distance(users_receipt,row)
                             for row in users_receipt.index
                            ]
```

[280]:
```
def get_counterfactual_distance(data,row):
    user_id = data['user_id'].loc[row]
    counterfactual_buyer = data['counterfactual_buyer'].loc[row]

    if ((user_id in user_buyer_distance.index) &
        (counterfactual_buyer in user_buyer_distance.columns)):
        output = user_buyer_distance[counterfactual_buyer].loc[user_id]
    else :
        output = np.nan
    return output
```

[281]:
```
users_receipt['counterfactual_distance'] =␣
↪[get_counterfactual_distance(users_receipt,row)
                                              for row in users_receipt.index
                                             ]
```

[282]:
```
user_buyer_distance.head()
```

[282]:

user_id	lr//nan//nan//1	lr//nan//nan//102	lr//nan//nan//104	\
100	32.819371	13248.011418	24.942663	
1000	53.379901	13212.817446	20.103252	
1002	47.400023	13238.403718	10.859041	
1004	53.553551	13212.771414	19.991087	
1006	51.822446	13235.860914	8.260469	

user_id	lr//nan//nan//105	lr//nan//nan//106	lr//nan//nan//107	\
100	53.316243	26.679406	8.381088	
1000	59.517287	56.221612	32.525556	
1002	40.735977	28.414093	7.844243	
1004	59.348067	56.192050	32.526995	
1006	37.315562	30.183422	12.231705	

user_id	lr//nan//nan//109	lr//nan//nan//112	lr//nan//nan//113	\
100	34.317010	65.829474	41.436333	

```
1000              6.355908          56.044285          45.371326
1002             29.976136          50.337231          27.471050
1004              6.581063          55.826154          45.207740
1006             30.459011          45.823785          23.644009

         lr//nan//nan//114  …  mill//76//199//nan  mill//8//191//nan  \
user_id                     …
100              40.783344  …           40.632112          38.259242
1000             47.576110  …           48.836304          32.390252
1002             27.681978  …           28.041537          44.390068
1004             47.422697  …           48.687886          32.613463
1006             24.243105  …           24.822301          47.236948

         mill//8//193//nan  mill//8//197//nan  mill//8//198//nan  \
user_id
100              49.010265          34.348542          48.471624
1000             36.980635          30.610300          36.982399
1002             53.796389          40.759999          53.431403
1004             37.202969          30.828904          37.205407
1006             56.189899          43.733861          55.871841

         mill//8//27//nan  mill//8//38//nan  mill//8//39//nan  \
user_id
100             36.467754         23.104722         37.851248
1000            31.374704         32.832156         32.195327
1002            42.658988         32.978098         44.012479
1004            31.596305         33.012594         32.418198
1006            45.549363         36.771086         46.872537

         mill//92//203//nan  mill//98//205//nan
user_id
100               56.388654           53.055187
1000              72.481894           67.928947
1002              48.262965           44.172875
1004              72.350161           67.793946
1006              46.353225           42.066089

[5 rows x 126 columns]
```

[283]:
```python
def get_counterfactual_distance_mean(data,row):
    #print (row)
    user_id = data['user_id'].loc[row]
    mask = old_buyers_df['user_id']==user_id
    buyer_list = list(set(list(old_buyers_df['buyer'][mask])) &
                      set(list(user_buyer_distance.columns)))

    if ((user_id in user_buyer_distance.index) & (len(buyer_list)>0)):
```

```
        output = np.nanmean(user_buyer_distance[buyer_list].loc[user_id])
    else :
        output = np.nan
    return output
```

[284]:
```
get_counterfactual_distance(users_receipt,120)
```

[284]:
```
15.98776171906593
```

[285]:
```
get_counterfactual_distance_mean(users_receipt,120)
```

[285]:
```
15.98776171906593
```

[286]:
```
users_receipt['counterfactual_distance_mean'] =␣
↪[get_counterfactual_distance_mean(users_receipt,row)
                                                    for row in users_receipt.index
                                                   ]
```

[287]:
```
users_receipt['counterfactual_distance_mean'].dropna().shape
```

[287]:
```
(5707,)
```

[288]:
```
users_receipt['delta_distance'] = (users_receipt['distance'] -
                                   users_receipt['counterfactual_distance']
                                  )
```

[289]:
```
users_receipt['post_grading_price_per_distance'] =␣
↪users_receipt['post_grading_price'] /users_receipt['distance']
```

[290]:
```
users_receipt['counterfactual_post_grading_price_per_distance'] =␣
↪(users_receipt['counterfactual_post_grading_price_mean'] /
                                                                  ␣
↪users_receipt['counterfactual_distance']
                                                                  )
```

[291]:
```
users_receipt['post_grading_price_per_distance_difference'] =␣
↪(users_receipt['post_grading_price_per_distance'] -
                                                                ␣
↪users_receipt['counterfactual_post_grading_price_per_distance']
                                                                )
```

3.1.14 Calculate upload order and daily upload order

[292]:
```

```

```
#Stop "cleaning the upload order to correct for multiple uploads on the same␣
 ↪day (see my comment on the fact that they are treated as single sales cycle␣
 ↪but two different observations -- let me know if this doesn't make sense or␣
 ↪if you have other suggestions"
def get_upload_order(row):
    #print(row)
    user_id = users_receipt['user_id'].loc[row]
    date = users_receipt['date'].loc[row]
    created_at = users_receipt['created_at'].loc[row]
    mask = (users_receipt['user_id'] == user_id)

    cols = ['date','user_id','created_at']
    tmp_df = users_receipt[cols][mask].drop_duplicates()
    tmp_df.sort_values(by=['date','created_at'],inplace=True)
    tmp_df['upload_order'] = 1

    for i in range(1,len(tmp_df)):
        if tmp_df['date'].iloc[i] == tmp_df['date'].iloc[i-1]:
            tmp_df['upload_order'].iloc[i] = tmp_df['upload_order'].iloc[i-1]
        else :
            tmp_df['upload_order'].iloc[i] = tmp_df['upload_order'].iloc[i-1] +␣
 ↪1

    mask_2 = ((tmp_df['created_at']==created_at))
    #print(tmp_df[mask_2].shape)
    output = tmp_df['upload_order'][mask_2].values[0]#.loc[row]
    return output
```

[293]:
```
mask = fruit_sale['user_id']=='201'

fruit_sale[mask]
```

[293]:
```
      Unnamed: 0  index fruit_sale_id                fruit_sale_identifier  \
4853        4853   4863         84253  93627b5e-3097-4a19-82a1-cd3646dc49fe

      telemetry_log_id buyer_type user_id  total_weight  total_sale  \
4853               NaN       ramp     201           NaN         NaN

     payment_date  …  ramp_id  do_id  ramp_manager_id  grade_fruit_sale  \
4853          NaN  …     69.0    NaN             61.0               NaN

      lr_id lrm_id     buyer_receipt             buyer  \
4853     69     61  lr//nan//nan//69  lr//nan//nan//69

     special_price_daily_index data_source
4853                       NaN  fruit_sale
```

```
[1 rows x 36 columns]
```

[294]:
```
users_receipt['upload_order'] = [get_upload_order(row) for row in
                                    users_receipt.index
                                   ]
```

[295]:
```
def get_daily_upload_order(row):
    #print(row)
    user_id = users_receipt['user_id'].loc[row]
    date = users_receipt['date'].loc[row]
    created_at = users_receipt['created_at'].loc[row]
    buyer_receipt = users_receipt['buyer_receipt'].loc[row]
    mask = ((users_receipt['user_id'] == user_id) &
            (users_receipt['date'] == date) &
            (users_receipt['buyer_receipt']==buyer_receipt)
           )

    cols = ['date','user_id','created_at','upload_order']

    tmp_df = users_receipt[cols][mask].drop_duplicates()
    tmp_df.sort_values(by=['upload_order'],inplace=True)
    tmp_df['daily_upload_order'] = list(range(1,len(tmp_df)+1))

    mask_2 = ((tmp_df['created_at']==created_at))
    #print(tmp_df)
    output = tmp_df['daily_upload_order'][mask_2].values[0]
    return output
```

[296]:
```
users_receipt['daily_upload_order'] = [get_daily_upload_order(row) for row in
                                       users_receipt.index
                                      ]
```

[297]:
```
def get_cumulative_new_buyer_fraction(row):
    #print(row)
    user_id = users_receipt['user_id'].loc[row]
    #date = users_receipt['date'].loc[row]
    #created_at = users_receipt['created_at'].loc[row]
    upload_order = users_receipt['upload_order'].loc[row]
    mask = ((users_receipt['user_id'] == user_id)&
            (users_receipt['upload_order']<=upload_order)
           )

    cols = ['user_id','upload_order', 'new_buyer']
    tmp_df = users_receipt[cols][mask].drop_duplicates()
    #tmp_df.sort_values(by=['date','created_at'],inplace=True)
    #tmp_df['upload_order'] = list(range(1,len(tmp_df)+1))
```

```
    #mask_2 = ((tmp_df['created_at']==created_at))
    #print(tmp_df[mask_2].shape)
    output = tmp_df['new_buyer'].mean()#.loc[row]
    return output
```

[298]:
```
get_cumulative_new_buyer_fraction(559)
```

[298]:
```
0.9811320754716981
```

[299]:
```
users_receipt['cumulative_new_buyer_fraction'] =␣
↪[get_cumulative_new_buyer_fraction(row) for row in
                                                    users_receipt.index
                                                   ]
```

3.1.15 Calculate weight

[300]:
```
def get_weight(row):
    #print(row)
    user_id = users_receipt['user_id'].loc[row]
    date = users_receipt['date'].loc[row]
    created_at = users_receipt['created_at'].loc[row]
    daily_upload_order = users_receipt['daily_upload_order'].loc[row]
    buyer_receipt = users_receipt['buyer_receipt'].loc[row]

    mask = ((transactions_sql['user_id'] == user_id) &
            (transactions_sql['date'] == date) &
            (transactions_sql['buyer_receipt'] == buyer_receipt) &
            ~(transactions_sql['net_weight'].isna())
           )

    cols = ['date','user_id','created_at','net_weight']
    tmp_df = transactions_sql[cols][mask].drop_duplicates().dropna()
    tmp_df.sort_values(by=['date','created_at'],inplace=True)
    tmp_df['daily_upload_order'] = list(range(1,len(tmp_df)+1))

    mask_2 = ((tmp_df['daily_upload_order']==daily_upload_order))
    #print(daily_upload_order)
    num_weight = len(tmp_df)
    if tmp_df[mask_2].empty:
        output = np.nan
    else :
        #print(tmp_df)
        output = tmp_df['net_weight'][mask_2].values[0]#.loc[row]
    return [output,num_weight]
```

[301]:
```
users_receipt['weight_&_num_weight'] = [get_weight(row) for row in
                                        users_receipt.index
                                       ]

users_receipt['net_weight'] = [row[0] for row in␣
↪users_receipt['weight_&_num_weight']]

users_receipt['num_weight'] = [row[1] for row in␣
↪users_receipt['weight_&_num_weight']]
```

[302]:
```
users_receipt['net_weight'].describe()
```

[302]:
```
count      4440.000000
mean       5274.397748
std       18514.995787
min         250.000000
25%        2480.000000
50%        4650.000000
75%        6958.500000
max      992000.000000
Name: net_weight, dtype: float64
```

[303]:
```
mask = users_receipt['net_weight']>25000

users_receipt['net_weight'][mask] = np.nan
```

[304]:
```
users_receipt['post_grading_weight_final'] =␣
↪users_receipt['imputed_post_grading_weight']
```

[305]:
```
users_receipt['weight_source'] = np.nan

mask = ~users_receipt['imputed_post_grading_weight'].isna()

users_receipt['weight_source'][mask] = 'Manual Imputation'
```

[306]:
```
users_receipt['adjusted_weight'].
↪fillna(users_receipt['total_weight']*users_receipt['grade_imputed_final']/
↪100, inplace = True)
```

[307]:
```
users_receipt['post_grading_weight_fruit_sale'] = users_receipt['total_weight']␣
↪- users_receipt['adjusted_weight']
```

[308]:
```
users_receipt['post_grading_weight_fruit_sale'].describe()
```

[308]:
```
count     2326.000000
mean      3520.370638
std       2723.191332
```

```
min      -2645.000000
25%       1446.500000
50%       2458.000000
75%       5006.500000
max      19400.000000
Name: post_grading_weight_fruit_sale, dtype: float64
```

[309]:
```
mask = users_receipt['post_grading_weight_fruit_sale']<0

users_receipt['post_grading_weight_fruit_sale'][mask] =np.nan
```

[310]:
```
users_receipt['post_grading_weight_final'].
 ↪fillna(users_receipt['post_grading_weight_fruit_sale'], inplace = True)
```

[311]:
```
mask =((~users_receipt['post_grading_weight_fruit_sale'].isna()) &
       (users_receipt['imputed_post_grading_weight'].isna())
      )

users_receipt['weight_source'][mask] = 'fruit_sale'
```

[312]:
```
users_receipt['grade_imputed_final'].describe()
```

[312]:
```
count    7585.000000
mean        4.574641
std         1.101844
min         1.000000
25%         4.000000
50%         4.490000
75%         4.990000
max        14.810000
Name: grade_imputed_final, dtype: float64
```

[313]:
```
users_receipt['post_grading_weight'] =␣
 ↪users_receipt['net_weight']*(1-users_receipt['grade_imputed_final']/100)
```

[314]:
```
users_receipt['post_grading_weight_final'].
 ↪fillna(users_receipt['post_grading_weight'], inplace = True)
```

[315]:
```
mask = users_receipt['post_grading_weight_final']>50000

users_receipt['post_grading_weight_final'][mask]# = np.nan
```

[315]:
```
Series([], Name: post_grading_weight_final, dtype: float64)
```

[316]:
```
mask =((~users_receipt['post_grading_weight'].isna()) &
       (users_receipt['post_grading_weight_fruit_sale'].isna()) &
       (users_receipt['imputed_post_grading_weight'].isna())
```

```
    )

users_receipt['weight_source'][mask] = 'transactions_sql'
```

[317]:
```
users_receipt['post_grading_weight_final'].describe()
```

[317]:
```
count     7316.000000
mean      4335.919696
std       2735.775458
min         20.000000
25%       1919.255500
50%       3970.210000
75%       6301.500000
max      19400.000000
Name: post_grading_weight_final, dtype: float64
```

[318]:
```
users_receipt['net_weight_final'] = users_receipt['post_grading_weight_final']/
↪(1-users_receipt['grade_imputed_final']/100)
```

[319]:
```
users_receipt['net_weight'].describe()
```

[319]:
```
count     4436.000000
mean      4858.361136
std       2778.796329
min        250.000000
25%       2480.000000
50%       4650.000000
75%       6942.500000
max      20000.000000
Name: net_weight, dtype: float64
```

[320]:
```
users_receipt['net_weight_final'].fillna(users_receipt['net_weight'], inplace=␣
↪True)
```

[321]:
```
users_receipt['total_weight'].describe()
```

[321]:
```
count     2329.000000
mean      3686.423787
std       2838.001527
min        125.000000
25%       1500.000000
50%       2590.000000
75%       5233.000000
max      20000.000000
Name: total_weight, dtype: float64
```

[322]:
```
users_receipt['net_weight_final'].fillna(users_receipt['total_weight'],␣
↪inplace= True)
```

[ ]:
```
```

[323]:
```
mask = users_receipt['net_weight_final']>60000

users_receipt['net_weight_final'][mask] = np.nan
```

[324]:
```
users_receipt['post_grading_weight_final'].describe()
```

[324]:
```
count     7316.000000
mean      4335.919696
std       2735.775458
min         20.000000
25%       1919.255500
50%       3970.210000
75%       6301.500000
max      19400.000000
Name: post_grading_weight_final, dtype: float64
```

[325]:
```
users_receipt['net_weight_final'].describe()
```

[325]:
```
count     7274.000000
mean      4560.331315
std       2880.772486
min         22.222222
25%       2000.000000
50%       4170.000000
75%       6640.000000
max      20000.000000
Name: net_weight_final, dtype: float64
```

[327]:
```
users_receipt['revenue'] =␣
↪users_receipt['post_grading_weight_final']*users_receipt['price_imputed_final']

#users_receipt['counterfactual_revenue_max'] =␣
↪users_receipt['counterfactual_post_grading_price_max']*users_receipt['net_weight_final']

users_receipt['counterfactual_revenue_max'] =␣
↪users_receipt['counterfactual_price_max']*users_receipt['post_grading_weight_final']

users_receipt['counterfactual_revenue_mean'] =␣
↪users_receipt['counterfactual_price_mean']*users_receipt['post_grading_weight_final']

users_receipt['counterfactual_revenue_special_group_robust'] =␣
↪(users_receipt['post_grading_weight_final']*
```

```
↪users_receipt['counterfactual_price_special_group_robust']
                                                                )
```

[328]:
```
users_receipt['counterfactual_revenue_max'].dropna().shape
```

[328]: (5285,)

[329]:
```
users_receipt['counterfactual_revenue_mean'].dropna().shape
```

[329]: (5285,)

3.1.16 Calculate data point per user

[330]:
```
data_point_per_user = pd.pivot_table(users_receipt,
                                     index=['user_id','role'],
                                     values = 'date',
                                     aggfunc = 'count'
                                    )

data_point_per_user.columns = ['#_of_data_points']

data_point_per_user.reset_index(inplace= True)

data_point_per_user.head()
```

[330]:

	user_id	role	#_of_data_points
0	100	Middle Man	179
1	1000	Middle Man	137
2	1004	Middle Man	5
3	1011	Amprah Farmer	5
4	1015	Middle Man	1

[331]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(data_point_per_user,
                                                    on = ['user_id','role'],
                                                    how = 'left'
                                                   )


print(users_receipt.shape)
```

```
(7933, 137)
(7933, 138)
```

3.1.17 Merging user relationship type

[332]:
```
users_relationship_breakdown = pd.pivot_table(last_most_transaction_do_ramp,
                                              index = ['user_id','role'],
                                              columns = 'buyer_in_pempem',
                                              values = 'buyer',
                                              aggfunc=lambda x:len(x.unique())
                                             )

users_relationship_breakdown.columns = ['buyer_not_in_pempem','buyer_in_pempem']

users_relationship_breakdown.fillna(0, inplace = True)

users_relationship_breakdown.head()
```

[332]:

user_id	role	buyer_not_in_pempem	buyer_in_pempem
100	Middle Man	1.0	1.0
1000	Middle Man	0.0	1.0
1002	Amprah Farmer	0.0	2.0
1004	Middle Man	0.0	1.0
1006	Amprah Farmer	1.0	1.0

[333]:
```
users_relationship_breakdown['relationship_type'] = np.nan
```

[334]:
```
mask = users_relationship_breakdown['buyer_not_in_pempem'] ==0

users_relationship_breakdown['relationship_type'][mask] = 'All Buyers in PemPem'
```

[335]:
```
mask = users_relationship_breakdown['buyer_in_pempem'] ==0

users_relationship_breakdown['relationship_type'][mask] = 'All Buyers not in␣
↪PemPem'
```

[336]:
```
mask = ((users_relationship_breakdown['buyer_in_pempem'] >0) &
        (users_relationship_breakdown['buyer_not_in_pempem'] >0))

users_relationship_breakdown['relationship_type'][mask] = 'Some Buyers in␣
↪PemPem'
```

[337]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(users_relationship_breakdown,
                                                    on = ['user_id','role'],
                                                    how = 'left'
                                                   )
```

```
print(users_receipt.shape)
```

```
(7933, 138)
(7933, 141)
```

[338]:
```
cols = ['user_id', 'village_id']

print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols],
                                    on = ['user_id'],
                                    how = 'left'
                                    )

print(users_receipt.shape)
```

```
(7933, 141)
(7933, 142)
```

3.1.18 Merging Average Mill Price information

[339]:
```
counterfactual_price['trans_type'] = [row[:4] for row in␣
↪counterfactual_price['buyer']]
```

[340]:
```
mask = counterfactual_price['trans_type']=='mill'

average_price_mill = pd.pivot_table(counterfactual_price[mask],
                                    index= 'date',
                                    values = 'price_counterfactual_table',
                                    aggfunc = np.nanmean
                                   )

average_price_mill.columns = ['mean_daily_mill_price']
```

[341]:
```
avg_mean_daily_mill_price = average_price_mill['mean_daily_mill_price'].mean()

print(avg_mean_daily_mill_price)

average_price_mill['mean_daily_mill_price'] =␣
↪average_price_mill['mean_daily_mill_price'] - avg_mean_daily_mill_price
```

```
1467.9362011594442
```

[342]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(average_price_mill,
                                    on = 'date',
```

```
                                    how = 'left'
                                    )

print(users_receipt.shape)
```

```
(7933, 142)
(7933, 143)
```

3.1.19 Calculate User & Buyer Density

[343]:
```
def get_user_buyer_density(row):
    #print(row)

    date = users_receipt['date'].loc[row]

    mask_user = users['date'] <= pd.to_datetime(date)

    users_list = list(users['user_id'][mask_user])

    user_id = users_receipt['user_id'].loc[row]

    if user_id in user_to_user_distance.columns :
        output = {}
        mask_5km = ((user_to_user_distance[user_id] < 5)&
                    (user_to_user_distance.index.isin(users_list))
                   )

        output['user_density_5km'] =␣
↪len(user_to_user_distance[user_id][mask_5km])

        mask_10km = ((user_to_user_distance[user_id] < 10)&
                     (user_to_user_distance.index.isin(users_list))
                    )

        output['user_density_10km'] =␣
↪len(user_to_user_distance[user_id][mask_10km])

        mask_buyer = ((users['date'] <= pd.to_datetime(date))&
                      (users['role'].isin(['DO', 'RAMP MANAGER']))
                     )

        buyers_list = list(users['user_id'][mask_buyer])

        mask_5km_buyer = ((user_to_user_distance[user_id] < 5)&
                          (user_to_user_distance.index.isin(buyers_list))
                         )
```

```
        output['buyer_density_5km'] =␣
↪len(user_to_user_distance[user_id][mask_5km_buyer])

        mask_10km_buyer = ((user_to_user_distance[user_id] < 10)&
                           (user_to_user_distance.index.isin(buyers_list))
                          )

        output['buyer_density_10km'] =␣
↪len(user_to_user_distance[user_id][mask_10km_buyer])

    else:
        output = np.nan

    return output
```

[344]:
```
get_user_buyer_density(10)
```

[344]:
```
{'user_density_5km': 0,
 'user_density_10km': 0,
 'buyer_density_5km': 0,
 'buyer_density_10km': 0}
```

[345]:
```
users_receipt['user_buyer_density'] = [get_user_buyer_density(row) for row in␣
↪users_receipt.index]
```

[346]:
```
users_receipt['user_density_5km'] = np.nan
users_receipt['user_density_10km'] = np.nan
users_receipt['buyer_density_5km'] = np.nan
users_receipt['buyer_density_10km'] = np.nan
```

[347]:
```
for row in users_receipt.index :
    if pd.Series(users_receipt['user_buyer_density'].loc[row]).notna()[0]:
        users_receipt['user_density_5km'].loc[row] =␣
↪users_receipt['user_buyer_density'].loc[row]['user_density_5km']
        users_receipt['user_density_10km'].loc[row] =␣
↪users_receipt['user_buyer_density'].loc[row]['user_density_10km']
        users_receipt['buyer_density_5km'].loc[row] =␣
↪users_receipt['user_buyer_density'].loc[row]['buyer_density_5km']
        users_receipt['buyer_density_10km'].loc[row] =␣
↪users_receipt['user_buyer_density'].loc[row]['buyer_density_10km']
```

[348]:
```
mask = users_receipt['user_buyer_density'].notna()

(users_receipt['user_density_5km'][mask] <=␣
↪users_receipt['user_density_10km'][mask]).unique()
```

[348]:
```
array([ True])
```

[349]:
```
mask = users_receipt['user_buyer_density'].notna()

(users_receipt['buyer_density_5km'][mask] <=␣
 ↪users_receipt['buyer_density_10km'][mask]).unique()
```

[349]:
```
array([ True])
```

[350]:
```
def get_user_density_village(row):
    #print(row)

    date = users_receipt['date'].loc[row]

    village = users_receipt['kecamatan_desa'].loc[row]

    mask_user = ((users['date'] <= pd.to_datetime(date)) &
                 (users['kecamatan_desa'] == village)
                )

    users_list = list(users['user_id'][mask_user])

    if isnan(village):
        output = np.nan
    else:
        output = len(users_list)

    return output
```

[351]:
```
def get_buyer_density_village(row):
    #print(row)

    date = users_receipt['date'].loc[row]

    village = users_receipt['kecamatan_desa'].loc[row]

    mask_buyer = ((users['date'] <= pd.to_datetime(date)) &
                  (users['kecamatan_desa'] == village) &
                   (users['role'].isin(['Do', 'Ramp Manager']))
                 )

    buyers_list = list(users['user_id'][mask_buyer])

    if isnan(village):
        output = np.nan
    else:
        output = len(buyers_list)

    return output
```

[352]:
```
users_receipt['user_density_village'] = [get_user_density_village(row) for row␣
↪in users_receipt.index]

users_receipt['buyer_density_village'] = [get_buyer_density_village(row) for␣
↪row in users_receipt.index]
```

3.1.20 Merging number of active users

[353]:
```
users_receipt['user_kecamatan_desa'] = users_receipt['kecamatan_desa']
```

[354]:
```
users_receipt['user_kecamatan_desa'].replace({'KERITANG__PETALONGAN':
↪'KERITANG__SENCALANG',
                                              'UKUI__UKUI II':'UKUI__UKUI DUA',
                                              'UKUI__UKUI I':'UKUI__UKUI SATU',
                                              'RENGAT__KAMPUNG KOTA BESAR':
↪'RENGAT__KAMPUNG BESAR KOTA',
                                              'KEMPAS__KULIM JAYA':
↪'KEMPAS__PEKAN TUA',
                                              'KEMUNING__SEKAYAN':
↪'KEMUNING__KERITANG',
                                             }, inplace=True)
```

[355]:
```
unique_daily_active_user_df = pd.pivot_table(do_ramp_visit,
                                             index='date', values = 'user_id',
                                             aggfunc=lambda x: len(x.unique()))

unique_daily_active_user_df.columns = ['unique_daily_active_user']

unique_daily_active_user_df.reset_index(inplace=True)

unique_daily_active_user_df.head()
```

[355]:

	date	unique_daily_active_user
0	2016-01-01	1
1	2019-05-09	9
2	2019-05-10	9
3	2019-05-11	14
4	2019-05-12	17

[356]:
```
counterfactual_price.columns
```

[356]:
```
Index(['date', 'buyer', 'buyer_receipt', 'grade_cleaned', 'price_cleaned',
       'max_price_cleaned', 'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
```

```
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa',
       'trans_type'],
      dtype='object')
```

[357]:
```
print(counterfactual_price.shape)

counterfactual_price = counterfactual_price.merge(unique_daily_active_user_df.
 ↪drop_duplicates(),
                                    on ='date',how = 'left')

print(counterfactual_price.shape)
```

```
(103587, 19)
(103587, 20)
```

[358]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(unique_daily_active_user_df.
 ↪drop_duplicates(),
                                    on ='date',how = 'left')

print(users_receipt.shape)
```

```
(7933, 151)
(7933, 152)
```

[359]:
```
users_receipt['kecamatan_desa']
```

[359]:
```
0          BATANG GANSAL__BELIMBING
1          BATANG GANSAL__BELIMBING
2          BATANG GANSAL__BELIMBING
3          BATANG GANSAL__BELIMBING
4            BATANG GANSAL__SIAMBUL
                    …
7928      RENGAT BARAT__TANI MAKMUR
7929      RENGAT BARAT__TANI MAKMUR
7930           KERITANG__SENCALANG
7931         SEBERIDA__TITIAN RESAK
7932    BATANG GANSAL__DANAU RAMBAI
Name: kecamatan_desa, Length: 7933, dtype: object
```

[360]:
```
connectedness_village_df.head()
```

[360]:
```
            buyer_kecamatan_desa          kecamatan_desa  \
0         PANGKALAN KURAS__PALAS  PANGKALAN KURAS__PALAS
1                 UKUI__AIR EMAS  PANGKALAN KURAS__PALAS
```

```
2  BANDAR PETALANGAN__AIR TERJUN  PANGKALAN KURAS__PALAS
3     BANDAR PETALANGAN__ANGKASA  PANGKALAN KURAS__PALAS
4              BUNUT__BAGAN LAGUH  PANGKALAN KURAS__PALAS

   kecamatan_desa_connected  same_village     user_kecamatan_desa  \
0                         0          True  PANGKALAN KURAS__PALAS
1                         0         False  PANGKALAN KURAS__PALAS
2                         0         False  PANGKALAN KURAS__PALAS
3                         0         False  PANGKALAN KURAS__PALAS
4                         1         False  PANGKALAN KURAS__PALAS

   shortest_path_length
0                   0.0
1                   6.0
2                   5.0
3                   3.0
4                   1.0
```

[361]:
```
print(users_receipt.shape)

cols = ['buyer_kecamatan_desa','user_kecamatan_desa','shortest_path_length']

users_receipt = users_receipt.merge(connectedness_village_df[cols],
                                    on =␣
 ↪['buyer_kecamatan_desa','user_kecamatan_desa'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 152)
(7933, 153)
```

[362]:
```
users_receipt['shortest_path_length'].describe()
```

[362]:
```
count    5862.000000
mean        1.842204
std         1.538398
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max        12.000000
Name: shortest_path_length, dtype: float64
```

[363]:
```
np.quantile(users_receipt['shortest_path_length'].dropna(),0.8)
```

[363]: 3.0

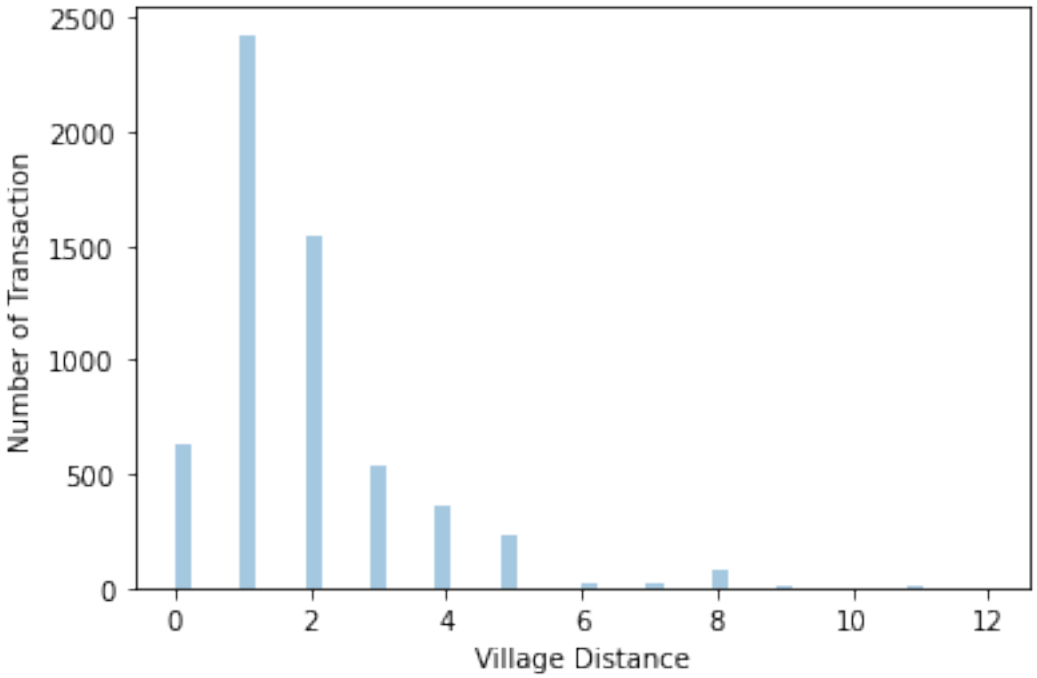
[364]:
```
np.quantile(users_receipt['shortest_path_length'].dropna(),0.9)
```

[364]: 4.0

[365]:
```
sns.distplot(users_receipt['shortest_path_length'].dropna(), kde=False)

plt.xlabel('Village Distance')
plt.ylabel('Number of Transaction')
```

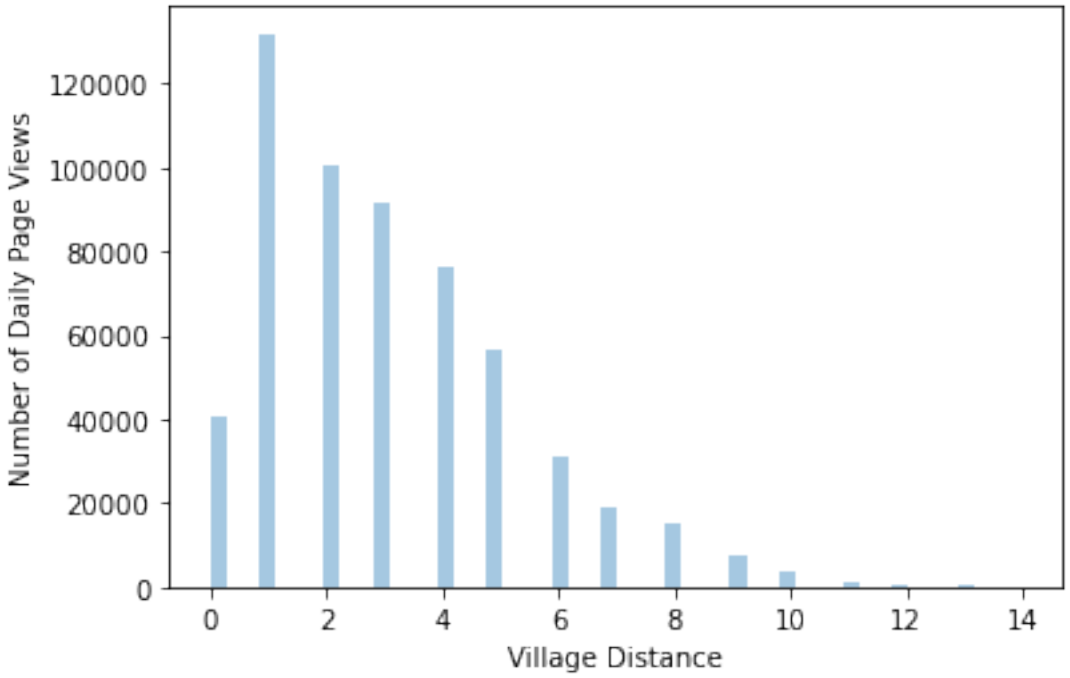
```
C:\Users\AK\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[365]: Text(0, 0.5, 'Number of Transaction')



[366]:
```
sns.distplot(do_ramp_visit['shortest_path_length'].dropna(), kde=False)

plt.xlabel('Village Distance')
plt.ylabel('Number of Daily Page Views')
```

[366]: Text(0, 0.5, 'Number of Daily Page Views')



[367]:
```
np.quantile(do_ramp_visit['shortest_path_length'].dropna(),0.86)
```

[367]: 5.0

[368]:
```
np.quantile(users_receipt['shortest_path_length'].dropna(),0.95)
```

[368]: 5.0

[369]:
```
np.quantile(users_receipt['shortest_path_length'].dropna(),0.97)
```

[369]: 5.0

[370]:
```
'''
village_neighbors_df = pd.DataFrame(connectedness_village_df['kecamatan_desa'].
↪drop_duplicates())

village_neighbors_df.reset_index(inplace=True)

village_neighbors_df.drop(columns=['index'], inplace=True)

village_neighbors_df.head()
'''
```

[370]:
```
"\nvillage_neighbors_df = pd.DataFrame(connectedness_village_df['kecamatan_desa'
].drop_duplicates())\n\nvillage_neighbors_df.reset_index(inplace=True)\n\nvillag
e_neighbors_df.drop(columns=['index'],
inplace=True)\n\nvillage_neighbors_df.head() \n"
```

[371]:
```
'''
def get_village_neighbors(num_village_threshold, row):
    source = village_neighbors_df['kecamatan_desa'].loc[row]
    mask = ((connectedness_village_df['shortest_path_length']<=num_village_threshold) &
            (connectedness_village_df['kecamatan_desa']==source)
           )

    output = list(connectedness_village_df['buyer_kecamatan_desa'][mask])

    return output
'''
```

[371]:
```
"\ndef get_village_neighbors(num_village_threshold, row):\n    source =
village_neighbors_df['kecamatan_desa'].loc[row]\n    mask =
((connectedness_village_df['shortest_path_length'] <=num_village_threshold) &\n
(connectedness_village_df['kecamatan_desa']==source)\n           )\n    \n
output = list(connectedness_village_df['buyer_kecamatan_desa'][mask])\n    \n
return output \n"
```

[372]:
```
'''
village_neighbors_df['village_neighbors_<=5'] = [get_village_neighbors(5, row) for
                                                 row in village_neighbors_df.index
                                                ]
'''
```

[372]:
```
"\nvillage_neighbors_df['village_neighbors_<=5'] = [get_village_neighbors(5,
row) for \n                                                 row in
village_neighbors_df.index\n                                                ]\n"
```

[373]:
```
village_neighbors_df.columns
```

[373]:
```
Index(['kecamatan_desa', 'user_kecamatan_desa', 'village_neighbors_<=5',
       'village_neighbors_<=8'],
      dtype='object')
```

[374]:
```
print(users_receipt.shape)

cols = ['village_neighbors_<=5', 'user_kecamatan_desa']
```

```
users_receipt = users_receipt.merge(village_neighbors_df[cols],
                                    on = ['user_kecamatan_desa'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 153)
(7933, 154)
```

[375]:
```
print(users_receipt.shape)

cols = ['village_neighbors_<=8', 'user_kecamatan_desa']

users_receipt = users_receipt.merge(village_neighbors_df[cols],
                                    on = ['user_kecamatan_desa'],
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 154)
(7933, 155)
```

[376]:
```
users_receipt.head()
```

[376]:

	id	trans_type	mill_id	lr_id	do_id	lrm_id	user_id	weight_receipt	\
0	10.0	mill	2	NaN	NaN	NaN	35	NaN	
1	21.0	mill	2	NaN	NaN	NaN	35	NaN	
2	35.0	mill	2	NaN	NaN	NaN	35	NaN	
3	41.0	mill	2	NaN	NaN	NaN	35	NaN	
4	46.0	mill	1	NaN	NaN	NaN	59	NaN	

	price_receipt	status	…	\
0	price_receipt/ef194e440719272468a5f7dd01658e5f…	1.0	…	
1	price_receipt/cd7b50a2767be467d9b7a8f30a674ba5…	1.0	…	
2	price_receipt/0420ef26302b0c20a3df325e56584442…	1.0	…	
3	price_receipt/f1daf27a617f97cb6a21173a6ff15f89…	1.0	…	
4	price_receipt/ef242808fdec83a09d25849d8edc6ec5…	1.0	…	

	user_density_10km	buyer_density_5km	buyer_density_10km	user_density_village	\
0	0.0	0.0	0.0	3.0	
1	0.0	0.0	0.0	3.0	
2	0.0	0.0	0.0	3.0	
3	0.0	0.0	0.0	4.0	
4	0.0	0.0	0.0	2.0	

```
   buyer_density_village       user_kecamatan_desa  unique_daily_active_user  \
0                    0.0  BATANG GANSAL__BELIMBING                       NaN
1                    0.0  BATANG GANSAL__BELIMBING                       NaN
2                    0.0  BATANG GANSAL__BELIMBING                       NaN
3                    0.0  BATANG GANSAL__BELIMBING                       NaN
4                    0.0    BATANG GANSAL__SIAMBUL                       NaN

  shortest_path_length                                village_neighbors_<=5  \
0                  NaN  [KERUMUTAN__MAK TEDUH, TELUK MERANTI__TELUK BI…
1                  NaN  [KERUMUTAN__MAK TEDUH, TELUK MERANTI__TELUK BI…
2                  NaN  [KERUMUTAN__MAK TEDUH, TELUK MERANTI__TELUK BI…
3                  NaN  [KERUMUTAN__MAK TEDUH, TELUK MERANTI__TELUK BI…
4                  NaN  [GAUNG__SIMPANG GAUNG, GAUNG__TELUK KABUNG, GA…

                               village_neighbors_<=8
0  [UKUI__AIR EMAS, BANDAR PETALANGAN__AIR TERJUN…
1  [UKUI__AIR EMAS, BANDAR PETALANGAN__AIR TERJUN…
2  [UKUI__AIR EMAS, BANDAR PETALANGAN__AIR TERJUN…
3  [UKUI__AIR EMAS, BANDAR PETALANGAN__AIR TERJUN…
4  [BANDAR PETALANGAN__AIR TERJUN, UKUI__BAGAN LI…

[5 rows x 155 columns]
```

[377]:
```
mask = village_neighbors_df['kecamatan_desa']=='KERITANG__PETALONGAN'

village_neighbors_df[mask]
```

[377]:
```
Empty DataFrame
Columns: [kecamatan_desa, user_kecamatan_desa, village_neighbors_<=5,
village_neighbors_<=8]
Index: []
```

[378]:
```
mask = users_receipt['village_neighbors_<=5'].isna()

list_village = users_receipt[['kecamatan_desa']][mask].dropna()

mask = village_neighbors_df['kecamatan_desa'].isin(list_village)

village_neighbors_df[mask]
```

[378]:
```
Empty DataFrame
Columns: [kecamatan_desa, user_kecamatan_desa, village_neighbors_<=5,
village_neighbors_<=8]
Index: []
```

[379]:
```
list_village.value_counts()
```

[379]:
```
Series([], dtype: int64)
```

[380]:
```
def get_num_local_price_available_5village(row):
    if isnan(users_receipt['village_neighbors_<=5'].loc[row]):
        output = np.nan
    else :
        buyer_kecamatan_desa_list = users_receipt['village_neighbors_<=5'].
↪loc[row]
        date = users_receipt['date'].loc[row]
        mask = ((counterfactual_price['buyer_kecamatan_desa'].
↪isin(buyer_kecamatan_desa_list)) &
                (counterfactual_price['date']==date)
               )
        output = len(counterfactual_price['price_counterfactual_table'][mask].
↪dropna())

    return output
```

[381]:
```
def get_num_local_price_available_8village(row):
    if isnan(users_receipt['village_neighbors_<=8'].loc[row]):
        output = np.nan
    else :
        buyer_kecamatan_desa_list = users_receipt['village_neighbors_<=8'].
↪loc[row]
        date = users_receipt['date'].loc[row]
        mask = ((counterfactual_price['buyer_kecamatan_desa'].
↪isin(buyer_kecamatan_desa_list)) &
                (counterfactual_price['date']==date)
               )
        output = len(counterfactual_price['price_counterfactual_table'][mask].
↪dropna())

    return output
```

[382]:
```
users_receipt['village_neighbors_<=5'].describe()
```

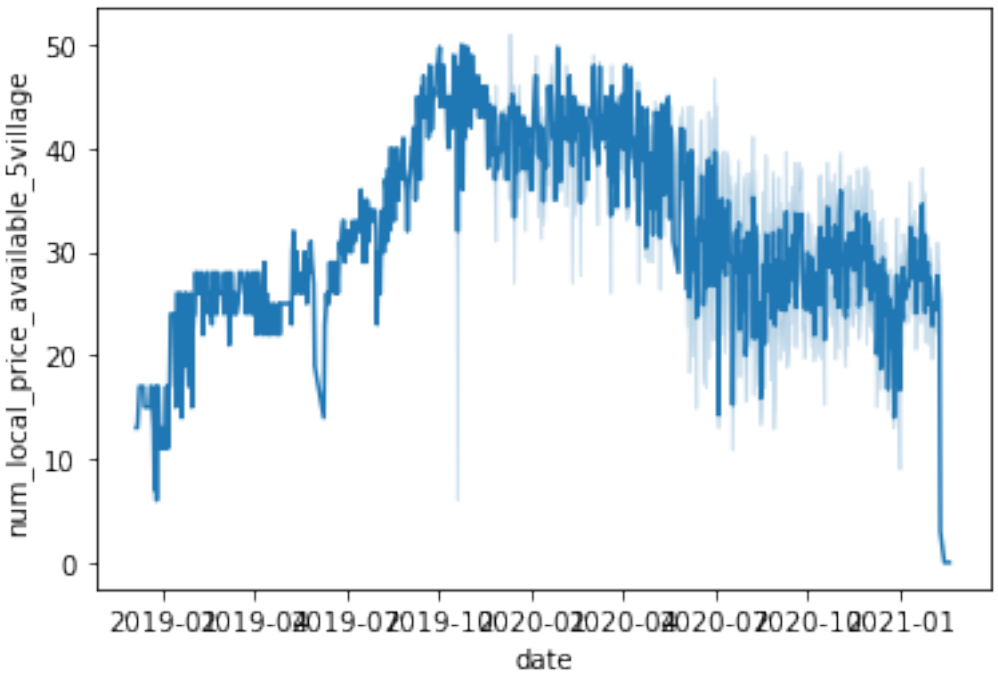
[382]:
```
count                                                  7535
unique                                                  142
top       [KERUMUTAN__MAK TEDUH, TELUK MERANTI__TELUK BI…
freq                                                   1463
Name: village_neighbors_<=5, dtype: object
```

[383]:
```
users_receipt['num_local_price_available_5village'] =␣
↪[get_num_local_price_available_5village(row) for
                                                       row in users_receipt.
↪index]
```

[384]:
```
users_receipt['num_local_price_available_8village'] =␣
↪[get_num_local_price_available_8village(row) for
                                                    row in users_receipt.
↪index]
```

[385]:
```
sns.lineplot(x ='date', y='num_local_price_available_5village', data =␣
↪users_receipt)
```

[385]: <AxesSubplot:xlabel='date', ylabel='num_local_price_available_5village'>



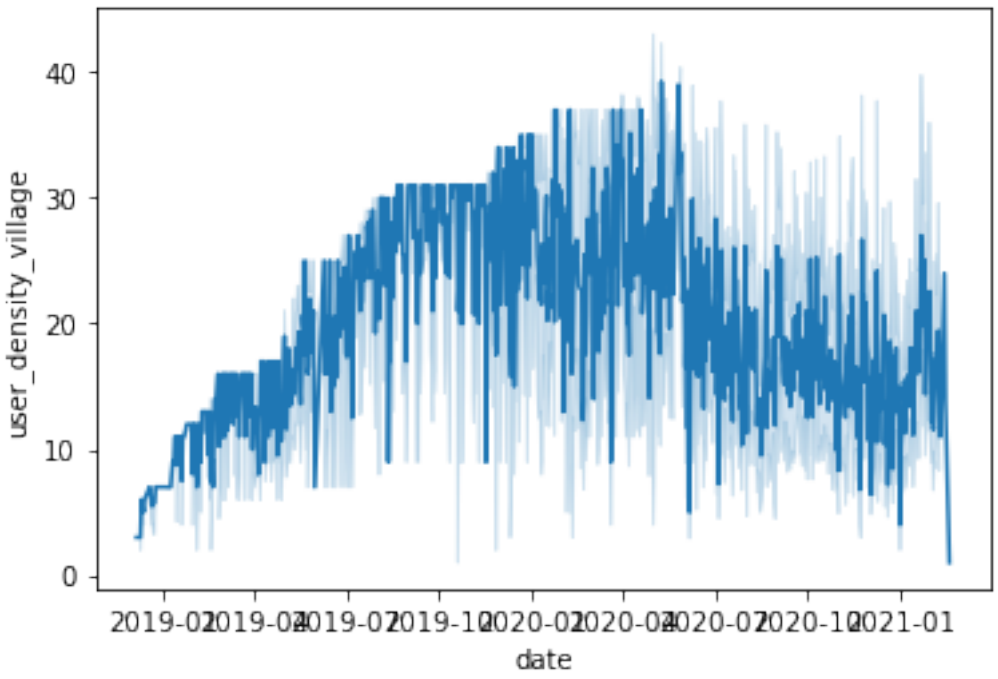
[386]:
```
users_receipt[['user_density_village','buyer_density_village','num_local_price_available_5vill
↪corr()
```

[386]:
```
                                    user_density_village  \
user_density_village                            1.000000
buyer_density_village                           0.339279
num_local_price_available_5village              0.521438

                                    buyer_density_village  \
user_density_village                             0.339279
buyer_density_village                            1.000000
num_local_price_available_5village               0.129141

                                    num_local_price_available_5village
user_density_village                                          0.521438
```

```
buyer_density_village                                  0.129141
num_local_price_available_5village                     1.000000
```

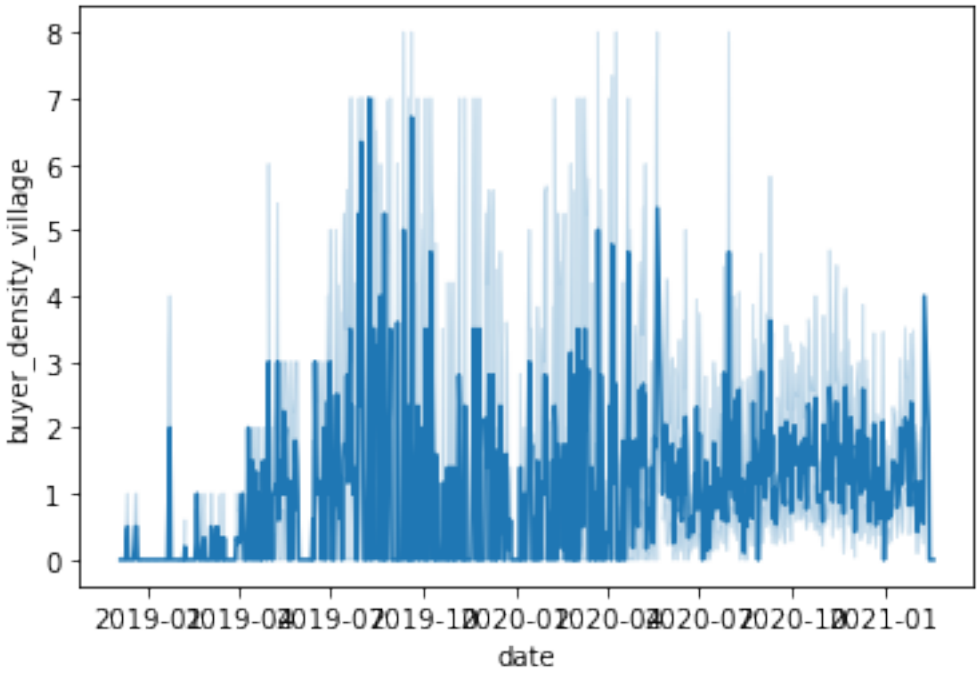
[387]:
```
sns.lineplot(x ='date', y='user_density_village', data = users_receipt)
```

[387]:
```
<AxesSubplot:xlabel='date', ylabel='user_density_village'>
```



[388]:
```
sns.lineplot(x ='date', y='buyer_density_village', data = users_receipt)
```

[388]:
```
<AxesSubplot:xlabel='date', ylabel='buyer_density_village'>
```



3.2 Create users level data

[389]:
```
old_buyer_per_user = pd.pivot_table(last_most_transaction_do_ramp,
                                    index = 'user_id',
                                    values='buyer',
                                    aggfunc= lambda x:len(x.unique())
                                   )

old_buyer_per_user.columns = ['old_buyers_per_user']
```

[390]:
```
print(users.shape)

users = users.merge(old_buyer_per_user,
                    on = 'user_id',
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 42)
(2060, 43)
```

[391]:
```
print(users.shape)

users = users.merge(data_point_per_user,
```

```
                    on = ['user_id','role'],
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 43)
(2060, 44)
```

[392]:
```
total_unique_buyer_transacted = pd.pivot_table(users_receipt,
                                               index = 'user_id',
                                               values='buyer',
                                               aggfunc = lambda x:len(x.
 ↪unique())
                                              )

total_unique_buyer_transacted.columns = ['total_buyer_transacted']

total_unique_buyer_transacted.head()
```

[392]:

user_id	total_buyer_transacted
100	7
1000	4
1004	2
1011	2
1015	1

[393]:
```
print(users.shape)

users = users.merge(total_unique_buyer_transacted,
                    on = ['user_id'],
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 44)
(2060, 45)
```

[394]:
```
mask = users_receipt['new_buyer']==True

total_unique_new_buyer_transacted = pd.pivot_table(users_receipt[mask],
                                               index = 'user_id',
                                               values='buyer',
                                               aggfunc = lambda x:len(x.
 ↪unique())
```

```
                                                )

total_unique_new_buyer_transacted.columns = ['total_new_buyer_transacted']

total_unique_new_buyer_transacted.head()
```

[394]:
```
         total_new_buyer_transacted
user_id
100                               6
1000                              2
1004                              1
1018                              1
1019                              6
```

[395]:
```
print(users.shape)

users = users.merge(total_unique_new_buyer_transacted,
                    on = ['user_id'],
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 45)
(2060, 46)
```

[396]:
```
old_buyer_breakdown = pd.pivot_table(last_most_transaction_do_ramp,
                                     index='user_id',
                                     columns = 'buyer_in_pempem',
                                     values = 'buyer',
                                     aggfunc = lambda x:len(x.unique())
                                    )

old_buyer_breakdown.columns = ['buyer_not_in_pempem','buyer_in_pempem']

old_buyer_breakdown.fillna(0,inplace=True)

old_buyer_breakdown.head()
```

[396]:
```
         buyer_not_in_pempem  buyer_in_pempem
user_id
100                      1.0              1.0
1000                     0.0              1.0
1002                     0.0              2.0
1004                     0.0              1.0
1006                     1.0              1.0
```

[397]:
```
print(users.shape)

users = users.merge(old_buyer_breakdown,
                    on = ['user_id'],
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 46)
(2060, 48)
```

[398]:
```
mask = users['old_buyers_per_user'] ==1

users_1_old_buyer = list(users['user_id'][mask].unique())
```

[399]:
```
print(users.shape)

cols = ['user_id','app_version']

users = users.merge(app_user_extra_infos[cols].
 ↪drop_duplicates(subset=['user_id'],keep='last'),
                    on = ['user_id'],
                    how = 'left'
                   )

print(users.shape)
```

```
(2060, 48)
(2060, 49)
```

[400]:
```
users['special_price_member'] = users['user_id'].
 ↪isin(price_group_user['user_id'])
```

[401]:
```
#users_receipt.drop(columns='app_version', inplace=True)

cols = ['user_id', 'old_buyers_per_user', 'total_buyer_transacted',
        'total_new_buyer_transacted', 'special_price_member'
       ]

print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols],
                                    on = 'user_id',
                                    how = 'left'
                                   )
```

```
print(users_receipt.shape)
```

```
(7933, 157)
(7933, 161)
```

[402]:
```
print(users_receipt.shape)

cols = ['user_id', 'all_hours_footprint', 'night_time_footprint',
       'night_time_footprint_type', 'night_time_footprint_equiv_radius',
       'night_time_footprint_area', 'all_hours_footprint_type',
       'all_hours_footprint_area', 'all_hours_footprint_equiv_radius',
       'night_time_modus_coord', 'all_hours_modus_coord', 'modus_coords',
       'modus_long', 'modus_lat']

users_receipt = users_receipt.merge(users_location,
                                    on = 'user_id',
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 161)
(7933, 175)
```

3.2.1 Group Membership

[403]:
```
group_membership = pd.pivot_table(price_group_user, index='user_id',
                                  columns = 'price_group_id', values =␣
 ↪'price_group_user_id',
                                  aggfunc='count')

group_membership.fillna(0, inplace = True)

group_membership.columns = ['dummy_special_group_' + str(col) for col in␣
 ↪group_membership.columns]

#group_membership['total_membership'] = group_membership.sum(axis=1)

group_membership.head()
```

[403]:

user_id	dummy_special_group_1	dummy_special_group_2	dummy_special_group_3	\
100	0.0	0.0	0.0	
1000	0.0	0.0	0.0	
1001	0.0	0.0	0.0	
1002	0.0	0.0	0.0	
1003	0.0	0.0	0.0	

```
         dummy_special_group_4  dummy_special_group_5  dummy_special_group_6  \
user_id
100                        0.0                    0.0                    0.0
1000                       0.0                    0.0                    0.0
1001                       0.0                    0.0                    0.0
1002                       0.0                    0.0                    0.0
1003                       0.0                    0.0                    0.0

         dummy_special_group_7  dummy_special_group_8  dummy_special_group_9  \
user_id
100                        0.0                    0.0                    0.0
1000                       0.0                    0.0                    0.0
1001                       0.0                    0.0                    0.0
1002                       0.0                    0.0                    0.0
1003                       0.0                    0.0                    0.0

         dummy_special_group_10  …  dummy_special_group_92  \
user_id                          …
100                         0.0  …                     0.0
1000                        0.0  …                     0.0
1001                        0.0  …                     0.0
1002                        0.0  …                     0.0
1003                        0.0  …                     0.0

         dummy_special_group_93  dummy_special_group_94  \
user_id
100                         0.0                     0.0
1000                        0.0                     0.0
1001                        0.0                     0.0
1002                        0.0                     0.0
1003                        0.0                     0.0

         dummy_special_group_95  dummy_special_group_96  \
user_id
100                         0.0                     0.0
1000                        0.0                     0.0
1001                        0.0                     0.0
1002                        0.0                     0.0
1003                        0.0                     0.0

         dummy_special_group_97  dummy_special_group_98  \
user_id
100                         0.0                     0.0
1000                        0.0                     0.0
1001                        0.0                     0.0
1002                        0.0                     0.0
```

```
1003                          0.0                           0.0

         dummy_special_group_99  dummy_special_group_100  \
user_id
100                           0.0                      0.0
1000                          0.0                      0.0
1001                          0.0                      0.0
1002                          0.0                      0.0
1003                          0.0                      0.0

         dummy_special_group_101
user_id
100                          0.0
1000                         0.0
1001                         0.0
1002                         0.0
1003                         0.0

[5 rows x 101 columns]
```

[404]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(group_membership,
                                    left_on = 'user_id',
                                    right_index= True,
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 175)
(7933, 276)
```

[405]:
```
cols = [col for col in users_receipt if col.startswith('dummy_special_group')]

users_receipt[cols].fillna(0, inplace = True)
```

[406]:
```
cols = list(group_membership.columns)

users_receipt[cols].fillna(0, inplace = True)

users_receipt['total_group_membership'] = users_receipt[cols].sum(axis=1)
```

3.2.2 Merging nighttime data

[407]:
```
print(users_receipt.shape)

cols = ['kecamatan_desa','nighttime_light','nighttime_light_density']

users_receipt = users_receipt.merge(relevant_village[cols],
                                    on = 'kecamatan_desa',
                                    how = 'left'
                                   )

print(users_receipt.shape)
```

```
(7933, 277)
(7933, 279)
```

4 Export tables

[408]:
```
output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 5 output/
 ↪'
```

[409]:
```
users_receipt.shape
```

[409]:
```
(7933, 279)
```

[410]:
```
users_receipt.to_excel(f"{output_path}" + "users_receipt_complete.xlsx")
counterfactual_price.to_excel(f"{output_path}" +"counterfactual_price_complete.
 ↪xlsx")
users.to_excel(f"{output_path}" + "users_complete.xlsx")
do_ramp_visit_daily.to_excel(f"{output_path}" +"do_ramp_visit_daily_complete.
 ↪xlsx")
old_buyers_df.to_excel(f"{output_path}" + "old_buyers_df_complete.xlsx")
```

[411]:
```
do_ramp_visit.to_excel(f"{output_path}" +"do_ramp_visit_complete.xlsx")
```

[ ]:
```

```

Appendix C

Reproduction Code : Notebook 6

Regression Analysis

Notebook 6 Regression Analysis

August 29, 2021

1 Importing Library

[1]:
```
import pandas as pd
import numpy as np
import scipy
from numpy import nan
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from bokeh.plotting import figure
from bokeh.io import output_file, show, push_notebook, output_notebook
from bokeh.models import ColumnDataSource, Select
from ipywidgets import interact
import scipy.stats as st
import sys
import geopandas as gpd
import descartes
from shapely.geometry import Point, Polygon
import random
from scipy import stats
import statsmodels.api as sm
import folium
import json
import os
from IPython.display import HTML, display
#import geopandas as gpd
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from numpy import array
import matplotlib.dates as mdates
from statsmodels.distributions.empirical_distribution import ECDF
import geopy.distance
from scipy.spatial import distance_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import networkx as nx
```

```
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
from sklearn.cluster import KMeans
from sklearn.neighbors import␣
↪(NeighborhoodComponentsAnalysis,KNeighborsClassifier)
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from community import community_louvain
from scipy import stats
import geopy.distance
from scipy.spatial import distance_matrix
import math

import esda
import libpysal as lps

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error
from sklearn.metrics.scorer import make_scorer
from sklearn import linear_model

from sklearn.metrics import classification_report,confusion_matrix
#from __future__ import division
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import max_error
from sklearn.metrics.scorer import make_scorer
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.dummy import DummyRegressor
import xgboost
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import explained_variance_score
from sklearn.linear_model import Lasso, LogisticRegression,LinearRegression
from sklearn import linear_model
from sklearn.feature_selection import SelectFromModel

import econml
from econml.ortho_forest import DiscreteTreatmentOrthoForest as CausalForest
from econml.sklearn_extensions.linear_model import WeightedLassoCVWrapper,␣
 ↪WeightedLasso, WeightedLassoCV
from matplotlib.colors import LinearSegmentedColormap

pd.options.mode.chained_assignment = None

import warnings
warnings.filterwarnings('ignore')
#sns.set_theme(color_codes=True)
```

```
C:\Users\AK\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143:
FutureWarning: The sklearn.metrics.scorer module is  deprecated in version 0.22
and will be removed in version 0.24. The corresponding classes / functions
should instead be imported from sklearn.metrics. Anything that cannot be
imported from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
#sns.set_theme(color_codes=True)
```

2 Importing data

[3]:
```
raw_data = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Raw Data/'

notebook_1_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 1␣
↪output/'

notebook_2_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 2␣
↪output/'

notebook_3_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 3␣
↪output/'

notebook_4_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 4␣
↪output/'

notebook_5_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 5␣
↪output/'

hte_analysis_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/HTE␣
↪Analysis Output/'

output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 6 output/
↪'

fig_output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 6␣
↪output/Figures/'
```

2.1 Importing users_receipt table

[4]:
```
users_receipt = pd.read_excel(f"{notebook_5_output}" + "users_receipt_complete.
↪xlsx")
users_receipt.drop(columns = ['Unnamed: 0'], inplace = True)

users_receipt['date'] = pd.to_datetime(users_receipt['date']).dt.date

users_receipt['user_id'] = users_receipt['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)
```

[5]:
```
users_receipt['price_source'].value_counts()
```

[5]:
```
Transactions_sql              3327
price fruit sale              2194
Manual Imputation             1213
Counterfactual price table     739
special_price_daily            276
```

```
Name: price_source, dtype: int64
```

[6]:
```
users_receipt['data_source']
```

[6]:
```
0               receipt
1               receipt
2               receipt
3               receipt
4               receipt
              …
7928    transactions_sql
7929    transactions_sql
7930    transactions_sql
7931    transactions_sql
7932    transactions_sql
Name: data_source, Length: 7933, dtype: object
```

[7]:
```
receipt_id_list = [46,'trx_86252','fs_630',432,21]

mask = users_receipt['receipt_id'].isin(receipt_id_list)

cols =␣
↪['receipt_id','data_source','imputed_price_manual','special_price','price_fruit_sale',
      ␣
↪'price_cleaned','imputed_price_counterfactual_table','price_imputed_final','price_source']

#users_receipt[cols][mask].to_excel(f"{output_path}" + "price_imputation_exampe.
↪xlsx")
```

[8]:
```
mask = users_receipt['price_source']=='special_price_daily'

cols = ['receipt_id','imputed_price_manual','special_price','price_fruit_sale',
      ␣
↪'price_cleaned','imputed_price_counterfactual_table','price_imputed_final']

users_receipt[cols][mask]
```

[8]:
```
     receipt_id  imputed_price_manual  special_price  price_fruit_sale  \
1914      17596                   NaN         1510.0               NaN
1925      17666                   NaN         1480.0               NaN
1932      17706                   NaN         1480.0               NaN
1955      17837                   NaN         1450.0               NaN
1963      17874                   NaN         1435.0               NaN
…           …                     …             …                  …
7723  trx_59453                   NaN         1510.0               NaN
7729  trx_60187                   NaN         1500.0               NaN
7742  trx_61657                   NaN         1405.0               NaN
```

```
7909  trx_80280                   NaN         1780.0               NaN
7931  trx_86252                   NaN         1970.0               NaN

      price_cleaned  imputed_price_counterfactual_table  price_imputed_final
1914         1595.0                              1552.5                1510.0
1925         1485.0                              1482.5                1480.0
1932         1485.0                              1485.0                1480.0
1955         1450.0                              1450.0                1450.0
1963         1440.0                              1440.0                1435.0
…               …                                   …                     …
7723         1510.0                              1510.0                1510.0
7729         1480.0                              1500.0                1500.0
7742         1425.0                              1415.0                1405.0
7909         1780.0                              1780.0                1780.0
7931         1970.0                              1970.0                1970.0

[276 rows x 7 columns]
```

[9]:
```
users_receipt['new_buyer_tuple'].value_counts()
```

[9]:
```
(False, 'False, buyer in old buyer list')      2718
(True, 'True, new ramp transaction')           1692
(True, 'True, not edge case')                  1351
(False, 'False, mill transaction no do_id')    1161
(True, 'True, new mill transaction')           1011
Name: new_buyer_tuple, dtype: int64
```

[10]:
```
users_receipt.shape
```

[10]:
```
(7933, 279)
```

[11]:
```
enddate = pd.to_datetime("2021-02-01")

mask = users_receipt['date'] < enddate

#users_receipt = users_receipt[mask]
```

[12]:
```
users_receipt['role'].replace({'Farmer':'Amprah Farmer'}, inplace = True)

users_receipt['platform_age']=(users_receipt['date'] - users_receipt['date'].
 ↪min()).dt.days

users_receipt['price_change_class'] = (np.
 ↪where(users_receipt['post_grading_price_difference']==0,0,
                                                 np.
 ↪where(users_receipt['post_grading_price_difference']>0,1,-1))
                                      )
```

```
users_receipt['delta_best_price_counterfactual'] = users_receipt['best_price']␣
↪- users_receipt['counterfactual_price_mean']

users_receipt['delta_best_price_viewed_counterfactual'] =␣
↪users_receipt['best_price_viewed'] -␣
↪users_receipt['counterfactual_price_mean']

#users_receipt['special_price_member'] = users_receipt['special_price_member']
```

[13]:
```
users_receipt['best_post_grading_price_viewed'].
↪fillna(users_receipt['post_grading_price'], inplace = True)

users_receipt['best_price_viewed'].fillna(users_receipt['price_imputed_final'],␣
↪inplace = True)
```

[14]:
```
(users_receipt['best_post_grading_price_viewed'] -␣
↪users_receipt['post_grading_price']).describe()
```

[14]:
```
count    7481.000000
mean       29.069004
std        51.726080
min         0.000000
25%         0.000000
50%         3.934500
75%        33.020500
max       458.712000
dtype: float64
```

[15]:
```
mask = (users_receipt['best_post_grading_price_viewed'] -␣
↪users_receipt['post_grading_price']) <0

users_receipt[['best_post_grading_price_viewed','post_grading_price']][mask]
```

[15]:
```
Empty DataFrame
Columns: [best_post_grading_price_viewed, post_grading_price]
Index: []
```

[16]:
```
mask = (users_receipt['best_post_grading_price'] -␣
↪users_receipt['post_grading_price']) <0

users_receipt[['best_post_grading_price','post_grading_price']][mask]
```

[16]:
```
Empty DataFrame
Columns: [best_post_grading_price, post_grading_price]
Index: []
```

[17]:
```
users_receipt['price_treatment_effect'] = users_receipt['price_imputed_final']␣
↪- users_receipt['counterfactual_price_mean']
```

[18]:
```
users_receipt['price_treatment_effect'].describe()
```

[18]:
```
count    5673.000000
mean        5.211565
std        37.348673
min      -543.750000
25%         0.000000
50%         0.000000
75%         0.000000
max       265.000000
Name: price_treatment_effect, dtype: float64
```

[19]:
```
users_receipt['post_grading_price_treatment_effect'] =␣
↪users_receipt['post_grading_price'] -␣
↪users_receipt['counterfactual_post_grading_price_mean']
```

2.2 Importing counterfactual_price table

[20]:
```
counterfactual_price = pd.
↪read_excel(f"{notebook_5_output}"+"counterfactual_price_complete.xlsx")

counterfactual_price.drop(columns = ['Unnamed: 0'], inplace =True)

counterfactual_price['date'] = pd.to_datetime(counterfactual_price['date']).dt.
↪date

counterfactual_price.head()
```

[20]:

	date	buyer	buyer_receipt	grade_cleaned	price_cleaned	max_price_cleaned	min_grade_cleaned	grade_cleaned_buyer_receipt	price_cleaned_buyer_receipt	max_price_cleaned_buyer_receipt
0	2018-12-01	lr//nan//nan//11	NaN	NaN	NaN	NaN	NaN	NaN		
1	2018-12-02	lr//nan//nan//11	NaN	NaN	NaN	NaN	NaN	NaN		
2	2018-12-03	lr//nan//nan//11	NaN	NaN	NaN	NaN	NaN	NaN		
3	2018-12-04	lr//nan//nan//11	NaN	NaN	NaN	NaN	NaN	NaN		
4	2018-12-05	lr//nan//nan//11	NaN	NaN	NaN	NaN	NaN	NaN		

```
0                           NaN                                NaN
1                           NaN                                NaN
2                           NaN                                NaN
3                           NaN                                NaN
4                           NaN                                NaN

   min_grade_cleaned_buyer_receipt  price_counterfactual_table  \
0                              NaN                         NaN
1                              NaN                         NaN
2                              NaN                         NaN
3                              NaN                         NaN
4                              NaN                         NaN

   grade_counterfactual_table  post_grading_price_counterfactual_table  \
0                         NaN                                      NaN
1                         NaN                                      NaN
2                         NaN                                      NaN
3                         NaN                                      NaN
4                         NaN                                      NaN

   max_price_counterfactual_table  min_grade_counterfactual_table  \
0                             NaN                             NaN
1                             NaN                             NaN
2                             NaN                             NaN
3                             NaN                             NaN
4                             NaN                             NaN

   max_post_grading_price_counterfactual_table buyer_kecamatan_desa  \
0                                          NaN                  NaN
1                                          NaN                  NaN
2                                          NaN                  NaN
3                                          NaN                  NaN
4                                          NaN                  NaN

  trans_type  unique_daily_active_user
0       lr//                      NaN
1       lr//                      NaN
2       lr//                      NaN
3       lr//                      NaN
4       lr//                      NaN
```

2.3 Importing users table

[21]:
```
users = pd.read_excel(f"{notebook_5_output}" + "users_complete.xlsx")
users.drop(columns = ['Unnamed: 0'], inplace = True)
users.replace({'NAN':np.nan}, inplace=True)
```

```
users['date'] = pd.to_datetime(users['date']).dt.date

users['user_id'] = users['user_id'].astype(str).replace('\.0', '', regex=True)
```

[22]:
```
users['role'].replace({'Farmer':'Amprah Farmer'}, inplace = True)
```

[23]:
```
mask  = users['kecamatan_desa'].notna()

users['kecamatan'] =np.nan

users['kecamatan'][mask] = [row[:row.index('__')] for row in␣
↪users['kecamatan_desa'][mask]]
```

2.4 Importing do_ramp_visit_daily table

[24]:
```
do_ramp_visit = pd.read_excel(f"{notebook_4_output}"+"do_ramp_visit_complete.
↪xlsx")
do_ramp_visit.drop(columns = ['Unnamed: 0'], inplace = True)

do_ramp_visit['date'] = pd.to_datetime(do_ramp_visit['date']).dt.date

do_ramp_visit['user_id'] = do_ramp_visit['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)
```

[25]:
```
do_ramp_visit_daily = pd.
↪read_excel(f"{notebook_5_output}"+"do_ramp_visit_daily_complete.xlsx")
do_ramp_visit_daily.drop(columns = ['Unnamed: 0'], inplace = True)

do_ramp_visit_daily['date'] = pd.to_datetime(do_ramp_visit_daily['date']).dt.
↪date

do_ramp_visit_daily['user_id'] = do_ramp_visit_daily['user_id'].astype(str).
↪replace('\.0', '', regex=True)
```

[26]:
```
do_ramp_visit_daily['role'].value_counts()
```

[26]:
```
Middle Man       10783
Amprah Farmer     4994
Ramp Manager      1394
Do                  38
Farmer               4
Name: role, dtype: int64
```

[27]:
```
do_ramp_visit_daily['role'].replace({'Farmer':'Amprah Farmer'}, inplace = True)
```

[28]:
```
do_ramp_visit_daily['role'].value_counts()
```

[28]:
```
Middle Man       10783
Amprah Farmer     4998
Ramp Manager      1394
Do                  38
Name: role, dtype: int64
```

[29]:
```
do_ramp_visit_daily.head()
```

[29]:

	user_id	role	date	membership_length	unique_old_buyer	unique_new_buyer	unique_total_buyer	%_old_buyer	no_old_buyer_view
0	100	Middle Man	2019-11-05	297	0	15	15	0.0	True
1	100	Middle Man	2019-11-07	299	0	6	6	0.0	True
2	100	Middle Man	2019-11-08	300	0	2	2	0.0	True
3	100	Middle Man	2019-11-09	301	0	12	12	0.0	True
4	100	Middle Man	2019-11-10	302	0	3	3	0.0	True

2.5 Importing relevant village table

[30]:
```
relevant_village = pd.
 ↪read_excel(f"{notebook_2_output}"+"relevant_village_complete.xlsx")

#geom_list = []

#for geom in relevant_village['geometry']:
#    try:
#        geom_list.append(shapely.wkt.loads(geom))
#    except:
#        geom_list.append(np.nan)

#relevant_village['geometry'] = geom_list

#mask = relevant_village['geometry'].notna()

#relevant_village = relevant_village[mask]

#relevant_village.reset_index(inplace = True)

relevant_village.drop(columns=['Unnamed: 0','index'], inplace=True)

relevant_village.head()
```

[30]:
```
   level_0  PROVNO  KABKOTNO  KECNO  DESANO    IDSP2010 PROVINSI     KABKOT  \
0        0      14         4     20      30  1404020030     RIAU  PELALAWAN
1        1      14         4     21      10  1404021010     RIAU  PELALAWAN
2        2      14         4     32      11  1404032011     RIAU  PELALAWAN
3        3      14         4     32       6  1404032006     RIAU  PELALAWAN
4        4      14         4     30      13  1404030013     RIAU  PELALAWAN

           KECAMATAN         DESA  SUMBER  \
0    PANGKALAN KURAS        PALAS  SP2010
1              UKUI     AIR EMAS  SP2010
2  BANDAR PETALANGAN   AIR TERJUN  SP2010
3  BANDAR PETALANGAN      ANGKASA  SP2010
4             BUNUT  BAGAN LAGUH  SP2010

                                            geometry  \
0  MULTIPOLYGON (((101.9055505690001 0.2754098930…
1  POLYGON ((102.1206216090001 -0.109291733999953…
2  POLYGON ((102.1467148460001 0.1343687620000651…
3  POLYGON ((102.125160625 0.189884060000054, 102…
4  POLYGON ((102.0921405430001 0.2523691220000615…

                 kecamatan_desa  relevant_village_index  nighttime_light  \
0        PANGKALAN KURAS__PALAS                     0.0        65.637956
1                UKUI__AIR EMAS                     1.0        43.561230
2  BANDAR PETALANGAN__AIR TERJUN                     2.0        39.424383
3    BANDAR PETALANGAN__ANGKASA                     3.0        15.253111
4            BUNUT__BAGAN LAGUH                     4.0       141.456076

         area  nighttime_light_density
0   38.116357                 1.722042
1   16.328344                 2.667829
2   34.678845                 1.136842
3   22.252859                 0.685445
4  127.280617                 1.111372
```

2.6 Importing users_location table

[31]:
```
users_location = pd.read_excel(f"{notebook_2_output}" + 'users_location.xlsx')

users_location.drop(columns = ['Unnamed: 0'],inplace = True)

users_location.head()
```

[31]:
```
   user_id                                all_hours_footprint  \
0       23  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1       24  LINESTRING (102.594475 -0.7465233, 102.6545117…
2       25  POLYGON ((102.5231233381282 -0.735018107855270…
```

```
3       27  POLYGON ((102.3894691781223 -0.640409193665700…
4       31  POLYGON ((102.511719261713 -0.7214787989871149…

                               night_time_footprint  \
0  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1                                                NaN
2  POLYGON ((102.5231568 -0.7350115, 102.5230697 …
3  POLYGON ((102.3458618 -0.6623945, 102.3449446 …
4  POLYGON ((102.4559663599486 -0.661479634417434…

                                           geometry  \
0  POLYGON ((102.5830556 -0.7890085, 102.5816209 …
1                                                NaN
2  POLYGON ((102.5231568 -0.7350115, 102.5230697 …
3  POLYGON ((102.3458618 -0.6623945, 102.3449446 …
4  POLYGON ((102.4559663599486 -0.661479634417434…

  night_time_footprint_type  night_time_footprint_equiv_radius  \
0                   Polygon                          10.345682
1                       NaN                                NaN
2                   Polygon                           0.105661
3                   Polygon                           5.502121
4                   Polygon                           3.801410

   night_time_footprint_area all_hours_footprint_type  \
0                 336.084051                  Polygon
1                        NaN               LineString
2                   0.035056                  Polygon
3                  95.058274                  Polygon
4                  45.375262                  Polygon

   all_hours_footprint_area  all_hours_footprint_equiv_radius  \
0                436.877202                         11.795460
1                       NaN                               NaN
2                  0.311361                          0.314896
3                112.991683                          5.998718
4                366.018211                         10.796588

  night_time_modus_coord  all_hours_modus_coord           modus_coords  \
0  (102.50811, -0.72163)  (102.50811, -0.72163)  (102.50811, -0.72163)
1                    NaN  (102.59448, -0.74652)  (102.59448, -0.74652)
2  (102.52397, -0.73443)    (102.52311, -0.735)  (102.52397, -0.73443)
3  (102.40708, -0.56157)  (102.40708, -0.56157)  (102.40708, -0.56157)
4   (102.41997, -0.6234)   (102.41997, -0.6234)   (102.41997, -0.6234)

   modus_long  modus_lat
0   102.50811   -0.72163
```

```
1   102.59448   -0.74652
2   102.52397   -0.73443
3   102.40708   -0.56157
4   102.41997   -0.62340
```

2.7 Importing old_buyers_df table

[32]:
```
old_buyers_df = pd.read_excel(f"{notebook_5_output}" + "old_buyers_df_complete.
↪xlsx")

old_buyers_df.drop(columns = ['Unnamed: 0'],inplace = True)

old_buyers_df['user_id'] = old_buyers_df['user_id'].astype(str).replace('\.0',␣
↪'', regex=True)

old_buyers_df.head()
```

[32]:

	user_id	buyer	buyer_receipt	source	trans_type
0	553	mill//1//21//nan	mill//1//nan//nan	in-app-survey	mill
1	554	mill//17//45//nan	mill//17//nan//nan	in-app-survey	mill
2	554	mill//33//185//nan	mill//33//nan//nan	in-app-survey	mill
3	555	mill//1//21//nan	mill//1//nan//nan	in-app-survey	mill
4	556	lr//nan//nan//45	lr//nan//nan//45	in-app-survey	lr

[33]:
```
old_buyers_df['source'].value_counts()
```

[33]:
```
in-app-survey               1251
special price membership     531
2018 survey                   45
Name: source, dtype: int64
```

[34]:
```
mask = old_buyers_df['source']=='2018 survey'

len(old_buyers_df['user_id'][mask].unique())
```

[34]:
```
17
```

[35]:
```
len(old_buyers_df['user_id'].unique())
```

[35]:
```
1181
```

2.8 Importing buyers table

[36]:
```
buyers = pd.read_excel(f"{notebook_4_output}"+'buyers_complete.xlsx')

buyers.drop(columns = ['Unnamed: 0'], inplace =True)
```

```
#buyers['date'] = pd.to_datetime(buyers['date']).dt.date

buyers['user_id'] = buyers['user_id'].astype(str).replace('\.0', '', regex=True)

buyers.head()
```

[36]:
```
   Unnamed: 0.1 user_id            buyer    buyer_receipt         lng  \
0             0     202  lr//nan//nan//1  lr//nan//nan//1  102.668298
1             1     203  lr//nan//nan//2  lr//nan//nan//2  102.514567
2             2     204  lr//nan//nan//3  lr//nan//nan//3  102.423992
3             3     205  lr//nan//nan//4  lr//nan//nan//4  102.412089
4             4     218  lr//nan//nan//5  lr//nan//nan//5  102.257117

        lat        buyer_kecamatan_desa counterfactual_buyer  \
0 -0.889027          KEMUNING__KERITANG      lr//nan//nan//1
1 -0.734371     BATANG GANSAL__SEBERIDA      lr//nan//nan//2
2 -0.628439   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//3
3 -0.551158   SEBERIDA__PANGKALAN KASAI      lr//nan//nan//4
4 -0.658824  BATANG CENAKU__BUKIT LIPAI      lr//nan//nan//5

  counterfactual_buyer_kecamatan_desa
0                  KEMUNING__KERITANG
1             BATANG GANSAL__SEBERIDA
2           SEBERIDA__PANGKALAN KASAI
3           SEBERIDA__PANGKALAN KASAI
4          BATANG CENAKU__BUKIT LIPAI
```

[37]:
```
mask  = buyers['buyer_kecamatan_desa'].notna()

buyers['buyer_kecamatan'] =np.nan

buyers['buyer_kecamatan'][mask] = [row[:row.index('__')] for row in␣
↪buyers['buyer_kecamatan_desa'][mask]]
```

[38]:
```
buyers['buyer_kecamatan_desa'].loc[0][:buyers['buyer_kecamatan_desa'].loc[0].
↪index('__')]
```

[38]:
```
'KEMUNING'
```

2.9 Importing village_df_hte

[39]:
```
village_df_hte = pd.read_csv(f"{hte_analysis_output}" + "village_df_hte.csv")
```

[40]:
```
village_df_hte['kecamatan_desa'] = [row[5:] for row in␣
↪village_df_hte['village_name']]
```

[41]:
```
village_df_hte.head()
```

[41]:
```
   Unnamed: 0  c_upload_order  c_mean_daily_mill_price  \
0           1              31               251.998009
1           2              31               251.998009
2           3              31               251.998009
3           4              31               251.998009
4           5              31               251.998009

   c_unique_daily_active_user  c_is_farmer  c_is_middleman  c_is_rampmanager  \
0                          86            0               1                 0
1                          86            0               1                 0
2                          86            0               1                 0
3                          86            0               1                 0
4                          86            0               1                 0

   c_special_price_member_dummy  c_coeff_var  \
0                             0     0.027509
1                             0     0.027509
2                             0     0.027509
3                             0     0.027509
4                             0     0.027509

   c_number_unique_price_shared_daily  …  m_md_2020-11  m_md_2020-12  \
0                                  28  …             0             0
1                                  28  …             0             0
2                                  28  …             0             0
3                                  28  …             0             0
4                                  28  …             0             0

   m_md_2021-01  m_md_2021-02   tau.hat  tau.variance  lower_90   upper_90  \
0             0             0  4.431777     27.636637 -4.216080  13.079634
1             0             0  4.420963     27.219178 -4.161331  13.003258
2             0             0  5.102762      9.530559  0.024383  10.181141
3             0             0  3.627851     22.029049 -4.092975  11.348677
4             0             0  4.420963     27.219178 -4.161331  13.003258

                        village_name                kecamatan_desa
0  v_vd_BATANG CENAKU__BUKIT LINGKAR  BATANG CENAKU__BUKIT LINGKAR
1    v_vd_BATANG CENAKU__KUALA KILAN    BATANG CENAKU__KUALA KILAN
2      v_vd_BATANG GANSAL__BELIMBING      BATANG GANSAL__BELIMBING
3   v_vd_BATANG GANSAL__DANAU RAMBAI   BATANG GANSAL__DANAU RAMBAI
4      v_vd_BATANG GANSAL__PANYAGUAN      BATANG GANSAL__PANYAGUAN

[5 rows x 156 columns]
```

[42]:
```
relevant_village.head()
```

[42]:
```
   level_0  PROVNO  KABKOTNO  KECNO  DESANO    IDSP2010 PROVINSI     KABKOT  \
0        0      14         4     20      30  1404020030     RIAU  PELALAWAN
1        1      14         4     21      10  1404021010     RIAU  PELALAWAN
2        2      14         4     32      11  1404032011     RIAU  PELALAWAN
3        3      14         4     32       6  1404032006     RIAU  PELALAWAN
4        4      14         4     30      13  1404030013     RIAU  PELALAWAN

           KECAMATAN         DESA  SUMBER  \
0    PANGKALAN KURAS        PALAS  SP2010
1              UKUI     AIR EMAS  SP2010
2  BANDAR PETALANGAN   AIR TERJUN  SP2010
3  BANDAR PETALANGAN      ANGKASA  SP2010
4             BUNUT  BAGAN LAGUH  SP2010

                                            geometry  \
0  MULTIPOLYGON (((101.9055505690001 0.2754098930…
1  POLYGON ((102.1206216090001 -0.109291733999953…
2  POLYGON ((102.1467148460001 0.1343687620000651…
3  POLYGON ((102.125160625 0.189884060000054, 102…
4  POLYGON ((102.0921405430001 0.2523691220000615…

                 kecamatan_desa  relevant_village_index  nighttime_light  \
0        PANGKALAN KURAS__PALAS                     0.0        65.637956
1                UKUI__AIR EMAS                     1.0        43.561230
2  BANDAR PETALANGAN__AIR TERJUN                     2.0        39.424383
3    BANDAR PETALANGAN__ANGKASA                     3.0        15.253111
4            BUNUT__BAGAN LAGUH                     4.0       141.456076

         area  nighttime_light_density
0   38.116357                 1.722042
1   16.328344                 2.667829
2   34.678845                 1.136842
3   22.252859                 0.685445
4  127.280617                 1.111372
```

[43]:
```
print(village_df_hte.shape)

cols =['geometry','nighttime_light_density','kecamatan_desa']

village_df_hte = village_df_hte.merge(relevant_village[cols], on =␣
 ↪'kecamatan_desa', how = 'left')

print(village_df_hte.shape)
```

```
(116, 156)
(116, 158)
```

[44]:
```
print(relevant_village.shape)

cols =['tau.hat','kecamatan_desa']

relevant_village = relevant_village.merge(village_df_hte[cols], on =␣
↪'kecamatan_desa', how = 'left')

print(relevant_village.shape)
```

```
(505, 17)
(505, 18)
```

[45]:
```
kecamatan_df_hte = pd.read_csv(f"{hte_analysis_output}" + "kecamatan_df_hte.
↪csv")
```

[46]:
```
kecamatan_df_hte['tau.hat.kecamatan'] = kecamatan_df_hte['tau.hat']
```

[47]:
```
kecamatan_df_hte['KECAMATAN'] = [row[5:].upper() for row in␣
↪kecamatan_df_hte['kecamatan_name']]
```

[48]:
```
kecamatan_df_hte['KECAMATAN'] = kecamatan_df_hte['KECAMATAN']
```

2.10 Importing indo_village

[49]:
```
indo_village = gpd.read_file(f"{raw_data}" + "indo_villages_sp2010.shp")

indo_village['kecamatan_desa'] = indo_village['KECAMATAN'] + '__' +␣
↪indo_village['DESA']
```

[50]:
```
mask = ((indo_village['PROVINSI']=='RIAU') &
        ((indo_village['KABKOT']=='INDRAGIRI HULU') |
         (indo_village['KABKOT']=='INDRAGIRI HILIR') |
         (indo_village['KABKOT']=='PELALAWAN')
        )
        )

indo_village = indo_village[mask]
```

[51]:
```
print(indo_village.shape)

cols =['tau.hat','kecamatan_desa']

indo_village = indo_village.merge(village_df_hte[cols], on = 'kecamatan_desa',␣
↪how = 'left')

print(indo_village.shape)
```

```
(505, 12)
(505, 13)
```

[52]:
```
indo_village.head()
```

[52]:
```
  PROVNO KABKOTNO KECNO DESANO    IDSP2010 PROVINSI     KABKOT  \
0     14       04   020    030  1404020030     RIAU  PELALAWAN
1     14       04   021    010  1404021010     RIAU  PELALAWAN
2     14       04   032    011  1404032011     RIAU  PELALAWAN
3     14       04   032    006  1404032006     RIAU  PELALAWAN
4     14       04   030    013  1404030013     RIAU  PELALAWAN

           KECAMATAN         DESA  SUMBER  \
0    PANGKALAN KURAS        PALAS  SP2010
1              UKUI     AIR EMAS  SP2010
2  BANDAR PETALANGAN   AIR TERJUN  SP2010
3  BANDAR PETALANGAN      ANGKASA  SP2010
4             BUNUT  BAGAN LAGUH  SP2010

                                            geometry  \
0  MULTIPOLYGON (((101.90555 0.27541, 101.91124 0…
1  POLYGON ((102.12062 -0.10929, 102.12062 -0.111…
2  POLYGON ((102.14671 0.13437, 102.14670 0.13423…
3  POLYGON ((102.12516 0.18988, 102.12410 0.18791…
4  POLYGON ((102.09214 0.25237, 102.09496 0.24955…

                 kecamatan_desa   tau.hat
0         PANGKALAN KURAS__PALAS       NaN
1                 UKUI__AIR EMAS  4.420963
2  BANDAR PETALANGAN__AIR TERJUN       NaN
3     BANDAR PETALANGAN__ANGKASA       NaN
4             BUNUT__BAGAN LAGUH       NaN
```

[53]:
```
print(indo_village.shape)

cols =['tau.hat.kecamatan','KECAMATAN']

indo_village = indo_village.merge(kecamatan_df_hte[cols], on = 'KECAMATAN', how␣
↪= 'left')

print(indo_village.shape)
```

```
(505, 13)
(505, 14)
```

3 Descriptive Statistics

[54]:
```
users_receipt['binary_role'] = np.where(users_receipt['role'].isin(['Amprah␣
↪Farmer','Farmer']),'farmer','non_farmer')
```

[55]:
```
users['binary_role'] = np.where(users['role'].isin(['Amprah␣
↪Farmer','Farmer']),'farmer','non_farmer')
```

[56]:
```
do_ramp_visit['binary_role'] = np.where(do_ramp_visit['role'].isin(['Amprah␣
↪Farmer','Farmer']),'farmer','non_farmer')
```

[57]:
```
do_ramp_visit_daily['binary_role'] = np.where(do_ramp_visit_daily['role'].
↪isin(['Amprah Farmer','Farmer']),
                                              'farmer','non_farmer')
```

3.1 Price imputation statistics

[58]:
```
users_receipt['price_source'].value_counts()
```

[58]:
```
Transactions_sql              3327
price fruit sale              2194
Manual Imputation             1213
Counterfactual price table     739
special_price_daily            276
Name: price_source, dtype: int64
```

[59]:
```
mask = ~users_receipt['price_cleaned'].isna()

users_receipt[mask].shape
```

[59]:
```
(4362, 286)
```

[60]:
```
mask = ~users_receipt['price_cleaned'].isna()

users_receipt['user_id'][mask].unique().shape
```

[60]:
```
(217,)
```

[61]:
```
mask = (~(users_receipt['imputed_price_counterfactual_table'].isna()) &
        (users_receipt['price_cleaned'].isna())
        )

users_receipt['imputed_price_counterfactual_table'][mask].shape
```

[61]:
```
(3273,)
```

[62]:
```
mask = (~(users_receipt['imputed_price_counterfactual_table'].isna()) |
         ~(users_receipt['price_cleaned'].isna())
        )

users_receipt[mask].shape
```

[62]:
```
(7635, 286)
```

[63]:
```
mask = (~(users_receipt['imputed_price_counterfactual_table'].isna()) |
         ~(users_receipt['price_cleaned'].isna())
        )

users_receipt['user_id'][mask].unique().shape
```

[63]:
```
(494,)
```

[64]:
```
users_receipt['imputed_price_manual'].dropna().shape
```

[64]:
```
(1213,)
```

[65]:
```
users_receipt['imputed_price_manual_counterfactual_table'].dropna().shape
```

[65]:
```
(1883,)
```

[66]:
```
users_receipt.columns
```

[66]:
```
Index(['id', 'trans_type', 'mill_id', 'lr_id', 'do_id', 'lrm_id', 'user_id',
       'weight_receipt', 'price_receipt', 'status',
       …
       'total_group_membership', 'nighttime_light', 'nighttime_light_density',
       'platform_age', 'price_change_class', 'delta_best_price_counterfactual',
       'delta_best_price_viewed_counterfactual', 'price_treatment_effect',
       'post_grading_price_treatment_effect', 'binary_role'],
      dtype='object', length=286)
```

[67]:
```
users_receipt['price_imputed_final'].dropna().shape
```

[67]:
```
(7749,)
```

[68]:
```
mask = ~users_receipt['price_imputed_final'].isna()

users_receipt['user_id'][mask].unique().shape
```

[68]:
```
(498,)
```

[69]:
```
mask = ((~users_receipt['price_imputed_final'].isna()) &
        (~users_receipt['counterfactual_price_mean'].isna())
```

```
       )

users_receipt[mask].shape
```

[69]: (5673, 286)

[70]:
```
mask = ((~users_receipt['price_imputed_final'].isna()) &
        (~users_receipt['counterfactual_price_mean'].isna())
       )

users_receipt['user_id'][mask].unique().shape
```

[70]: (391,)

3.2 Users statistics

[71]:
```
pd.pivot_table(users_receipt, index = 'binary_role', values = 'user_id',
               aggfunc=['count',lambda x:len(x.unique())],margins=True
              )
```

[71]:

	count	<lambda>
	user_id	user_id
binary_role		
farmer	1456	233
non_farmer	6477	280
All	7933	513

[72]:
```
users_statistics = []
```

[73]:
```
mask = ((users['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Data Points per User')

data_points_stats = pd.pivot_table(users[mask],
                                   index= 'binary_role',
                                   margins=True,
                                   values='#_of_data_points',
                                   aggfunc=[np.nanmean, np.nanmedian,np.nanmin,␣
 ↪np.nanmax,np.nanstd]
                                  )

data_points_stats.columns = ['Mean','Median','Min','Max','Std Dev']

data_points_stats['Statistics'] = 'Data Points per User'

data_points_stats.reset_index(inplace = True)
```

```
users_statistics.append(data_points_stats)

data_points_stats
```

Data Points per User

[73]:

	binary_role	Mean	Median	Min	Max	Std Dev	Statistics
0	farmer	6.248927	1.0	1.0	185.0	16.517175	Data Points per User
1	non_farmer	23.132143	2.0	1.0	419.0	59.190783	Data Points per User
2	All	15.463938	2.0	1.0	419.0	45.820090	Data Points per User

[74]:
```
mask = ((do_ramp_visit_daily['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Daily Unique Buyer Viewed')

unique_buyer_viewed_stats = pd.pivot_table(do_ramp_visit_daily[mask],
                                           index= 'binary_role',
                                           margins=True,
                                           values='unique_total_buyer',
                                           aggfunc=[np.nanmean, np.nanmedian,np.
 ↪nanmin, np.nanmax,np.nanstd]
                                          )

unique_buyer_viewed_stats.columns = ['Mean','Median','Min','Max','Std Dev']

unique_buyer_viewed_stats['Statistics'] = 'Daily Unique Buyers Viewed'

unique_buyer_viewed_stats.reset_index(inplace = True)

users_statistics.append(unique_buyer_viewed_stats)

unique_buyer_viewed_stats
```

Daily Unique Buyer Viewed

[74]:

	binary_role	Mean	Median	Min	Max	Std Dev	Statistics
0	farmer	8.380707	6	1	62	8.700533	Daily Unique Buyers Viewed
1	non_farmer	7.972176	5	1	74	8.678402	Daily Unique Buyers Viewed
2	All	8.075873	5	1	74	8.685245	Daily Unique Buyers Viewed

[75]:
```
mask = ((do_ramp_visit_daily['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Daily Unique New Buyer Viewed')

unique_new_buyer_viewed_stats = pd.pivot_table(do_ramp_visit_daily[mask],
                                               index= 'binary_role',
                                               margins=True,
                                               values='unique_new_buyer',
                                               aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                               )

unique_new_buyer_viewed_stats.columns = ['Mean','Median','Min','Max','Std Dev']

unique_new_buyer_viewed_stats['Statistics'] = 'Daily Unique New Buyers Viewed'

unique_new_buyer_viewed_stats.reset_index(inplace = True)

users_statistics.append(unique_new_buyer_viewed_stats)

unique_new_buyer_viewed_stats
```

```
Daily Unique New Buyer Viewed
```

[75]:

	binary_role	Mean	Median	Min	Max	Std Dev	\
0	farmer	7.686413	5	0	61	8.592147	
1	non_farmer	7.381124	4	0	74	8.513060	
2	All	7.458615	4	0	74	8.533647	

	Statistics
0	Daily Unique New Buyers Viewed
1	Daily Unique New Buyers Viewed
2	Daily Unique New Buyers Viewed

[76]:
```
mask = ((users['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Number of Old Buyer Reported')

old_buyers_reported_stats = pd.pivot_table(users[mask],
                                               index= 'binary_role',
                                               margins=True,
                                               values='old_buyers_per_user',
                                               aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                               )
```

```
old_buyers_reported_stats.columns = ['Mean','Median','Min','Max','Std Dev']

old_buyers_reported_stats['Statistics'] = 'Old Buyers per User'

old_buyers_reported_stats.reset_index(inplace = True)

users_statistics.append(old_buyers_reported_stats)

old_buyers_reported_stats
```

Number of Old Buyer Reported

[76]:

	binary_role	Mean	Median	Min	Max	Std Dev	Statistics
0	farmer	1.424893	1.0	1.0	6.0	0.703855	Old Buyers per User
1	non_farmer	1.821429	2.0	1.0	5.0	0.886116	Old Buyers per User
2	All	1.641326	1.0	1.0	6.0	0.830702	Old Buyers per User

[77]:
```
mask = ((users['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Unique Buyers Transacted per User')

total_unique_buyer_transacted_stats = pd.pivot_table(users[mask],
                                                     index= 'binary_role',
                                                     margins=True,
                                                     values='total_buyer_transacted',
                                                     aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                                     )

total_unique_buyer_transacted_stats.columns = ['Mean','Median','Min','Max','Std␣
 ↪Dev']

total_unique_buyer_transacted_stats['Statistics'] = 'Total Buyer Transacted'

total_unique_buyer_transacted_stats.reset_index(inplace = True)

users_statistics.append(total_unique_buyer_transacted_stats)

total_unique_buyer_transacted_stats
```

Unique Buyers Transacted per User

[77]:

	binary_role	Mean	Median	Min	Max	Std Dev	Statistics
0	farmer	1.532189	1.0	1.0	10.0	1.214156	Total Buyer Transacted
1	non_farmer	2.500000	1.0	1.0	28.0	2.908398	Total Buyer Transacted

```
2         All  2.060429     1.0  1.0  28.0  2.345053  Total Buyer Transacted
```

[78]:
```
mask = ((users['user_id'].isin(users_receipt['user_id'].unique()))
       )

print('Unique New Buyers Transacted per User')

total_unique_new_buyer_transacted_stats = pd.pivot_table(users[mask],
                                                         index= 'binary_role',
                                                         margins=True,
                                                        ␣
 ↪values='total_new_buyer_transacted',
                                                         aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                                         )

total_unique_new_buyer_transacted_stats.columns =␣
 ↪['Mean','Median','Min','Max','Std Dev']

total_unique_new_buyer_transacted_stats['Statistics'] = 'Total New Buyer␣
 ↪Transacted'

total_unique_new_buyer_transacted_stats.reset_index(inplace = True)

users_statistics.append(total_unique_new_buyer_transacted_stats)

total_unique_new_buyer_transacted_stats
```

```
Unique New Buyers Transacted per User
```

[78]:
```
   binary_role      Mean  Median  Min   Max   Std Dev  \
0       farmer  1.422764     1.0  1.0   6.0  0.975168
1   non_farmer  2.445026     1.0  1.0  25.0  2.882956
2          All  2.044586     1.0  1.0  25.0  2.376488

                    Statistics
0  Total New Buyer Transacted
1  Total New Buyer Transacted
2  Total New Buyer Transacted
```

[79]:
```
users_statistics_df = pd.concat(users_statistics, ignore_index=True)

users_statistics_df.head()
```

[79]:
```
   binary_role       Mean  Median  Min    Max    Std Dev  \
0       farmer   6.248927     1.0  1.0  185.0  16.517175
1   non_farmer  23.132143     2.0  1.0  419.0  59.190783
```

```
2         All  15.463938     2.0  1.0  419.0  45.820090
3      farmer   8.380707     6.0  1.0   62.0   8.700533
4  non_farmer   7.972176     5.0  1.0   74.0   8.678402

                   Statistics
0        Data Points per User
1        Data Points per User
2        Data Points per User
3  Daily Unique Buyers Viewed
4  Daily Unique Buyers Viewed
```

[80]:
```
users_statistics_df.columns = ['User Role', 'Mean', 'Median', 'Min', 'Max',␣
↪'Std Dev', 'Statistics']
```

[81]:
```
users_statistics_df['User Role'].replace({'All':'All Role',
                                          'farmer':'Farmer',
                                          'non_farmer':'Middle Man & Ramp␣
↪Manager'
                                         }, inplace = True)
```

[82]:
```
users_statistics_df['latex'] = ("\\" + "\\" +
                                users_statistics_df['Statistics'] +" & " +
                                users_statistics_df['User Role'] + " & " +
                                users_statistics_df['Mean'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                users_statistics_df['Median'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) + " & " +
                                users_statistics_df['Min'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                users_statistics_df['Max'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                users_statistics_df['Std Dev'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) +
                                "\\" + "\\"
                               )
```

[83]:
```
users_statistics_df.to_excel(f"{output_path}" + "users_statistics_df.xlsx")
```

3.3 Price statistics

[84]:
```
price_statistics_list = []
```

[85]:
```
counterfactual_price.columns
```

[85]:
```
Index(['date', 'buyer', 'buyer_receipt', 'grade_cleaned', 'price_cleaned',
       'max_price_cleaned', 'min_grade_cleaned', 'grade_cleaned_buyer_receipt',
```

```
       'price_cleaned_buyer_receipt', 'max_price_cleaned_buyer_receipt',
       'min_grade_cleaned_buyer_receipt', 'price_counterfactual_table',
       'grade_counterfactual_table', 'post_grading_price_counterfactual_table',
       'max_price_counterfactual_table', 'min_grade_counterfactual_table',
       'max_post_grading_price_counterfactual_table', 'buyer_kecamatan_desa',
       'trans_type', 'unique_daily_active_user'],
      dtype='object')
```

[86]:
```
#mask = (~(transactions_sql['Outlier'])
#       )

print('Price Statistics')

price_stats = pd.pivot_table(counterfactual_price,
                                                index= 'trans_type',
                                                margins=True,
                                                ␣
 ↪values='price_counterfactual_table',
                                                aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                                )

price_stats.columns = ['Mean','Median','Min','Max','Std Dev']

price_stats['Statistics'] = 'Price'

price_stats.reset_index(inplace = True)

price_statistics_list.append(price_stats)

price_stats
```

```
Price Statistics
```

[86]:

	trans_type	Mean	Median	Min	Max	Std Dev	Statistics
0	lr//	1525.499627	1560.0	780.0	2040.0	287.834274	Price
1	mill	1497.622339	1470.0	810.0	2200.0	280.336062	Price
2	All	1504.582584	1500.0	780.0	2200.0	282.475782	Price

[87]:
```
#mask = (~(transactions_sql['Outlier'])
#       )

print('Price Statistics')

price_stats = pd.pivot_table(users_receipt,
                                                index= 'trans_type',
                                                margins=True,
```

```
                                                values='price_imputed_final',
                                                aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                                )

price_stats.columns = ['Mean','Median','Min','Max','Std Dev']

price_stats['Statistics'] = 'Price'

price_stats.reset_index(inplace = True)

price_statistics_list.append(price_stats)

price_stats
```

Price Statistics

[87]:

	trans_type	Mean	Median	Min	Max	Std Dev	Statistics
0	lr	1720.014573	1800.0	1000.0	2050.0	208.536482	Price
1	mill	1599.826647	1630.0	905.0	2200.0	285.212637	Price
2	All	1647.721900	1700.0	905.0	2200.0	264.018511	Price

[88]:
```
#mask = (~(transactions_sql['Outlier'])
#       )

print('Corrected Price Statistics')

corrected_price_stats = pd.pivot_table(users_receipt,
                                                index= 'trans_type',
                                                margins=True,
                                                values='post_grading_price',
                                                aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                                )

corrected_price_stats.columns = ['Mean','Median','Min','Max','Std Dev']

corrected_price_stats['Statistics'] = 'Post Grading Price'

corrected_price_stats.reset_index(inplace = True)

price_statistics_list.append(corrected_price_stats)

corrected_price_stats
```

Corrected Price Statistics

[88]:
```
  trans_type         Mean     Median        Min      Max     Std Dev  \
0         lr  1653.256798  1713.6175  940.10000  1989.90  195.429099
1       mill  1525.058727  1548.3370  861.96725  2101.22  274.451307
2        All  1575.628505  1623.2480  861.96725  2101.22  254.139883

           Statistics
0  Post Grading Price
1  Post Grading Price
2  Post Grading Price
```

[89]:
```
scipy.stats.
↪ttest_ind(users_receipt['price_imputed_final'][users_receipt['trans_type']=='lr'].
↪dropna(),
                    ␣
↪users_receipt['price_imputed_final'][users_receipt['trans_type']=='mill'].
↪dropna()
                    )
```

[89]:
```
Ttest_indResult(statistic=20.12283565726371, pvalue=7.944176917152925e-88)
```

[90]:
```
mask = users_receipt['trans_type']=='lr'

users_receipt['buyer'][mask].value_counts()
```

[90]:
```
lr//nan//nan//69     434
lr//nan//nan//50     352
lr//nan//nan//47     196
lr//nan//nan//73     187
lr//nan//nan//2      186
                    …
lr//nan//nan//104      1
lr//nan//nan//109      1
lr//nan//nan//24       1
lr//nan//nan//99       1
lr//nan//nan//131      1
Name: buyer, Length: 69, dtype: int64
```

[91]:
```
mask = users_receipt['trans_type']=='mill'

len(users_receipt['buyer'][mask].value_counts())
```

[91]:
```
56
```

[92]:
```
#mask = (~(transactions_sql['Outlier'])
#       )

print('Counterfactual Price Statistics')
```

```
counterfactual_price_stats = pd.pivot_table(users_receipt,
                                            index= 'trans_type',
                                            margins=True,
                                           ␣
 ↪values='counterfactual_price_mean',
                                            aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                            )

counterfactual_price_stats.columns = ['Mean','Median','Min','Max','Std Dev']

counterfactual_price_stats['Statistics'] = 'counterfactual_price_mean'

counterfactual_price_stats.reset_index(inplace = True)

price_statistics_list.append(counterfactual_price_stats)

counterfactual_price_stats
```

```
Counterfactual Price Statistics
```

[92]:

	trans_type	Mean	Median	Min	Max	Std Dev	\
0	lr	1700.512534	1760.0	1030.0	2065.0	206.736687	
1	mill	1597.161193	1630.0	905.0	2200.0	280.326405	
2	All	1641.830298	1700.0	905.0	2200.0	256.303593	

	Statistics
0	counterfactual_price_mean
1	counterfactual_price_mean
2	counterfactual_price_mean

[93]:
```
#mask = (~(transactions_sql['Outlier'])
#       )

print('Counterfactual Price Corrected Statistics')

counterfactual_price_corrected_stats = pd.pivot_table(users_receipt,
                                            index= 'trans_type',
                                            margins=True,
                                           ␣
 ↪values='counterfactual_post_grading_price_mean',
                                            aggfunc=[np.nanmean, np.
 ↪nanmedian,np.nanmin, np.nanmax,np.nanstd]
                                            )
```

```
counterfactual_price_corrected_stats.columns =␣
↪['Mean','Median','Min','Max','Std Dev']

counterfactual_price_corrected_stats['Statistics'] =␣
↪'counterfactual_post_grading_price_mean'

counterfactual_price_corrected_stats.reset_index(inplace = True)

price_statistics_list.append(counterfactual_price_corrected_stats)

counterfactual_price_corrected_stats
```

Counterfactual Price Corrected Statistics

[93]:

	trans_type	Mean	Median	Min	Max	Std Dev	\
0	lr	1626.772090	1682.639000	973.59000	1969.50	196.474156	
1	mill	1521.894932	1547.397250	861.96725	2101.22	272.542686	
2	All	1567.034368	1619.661312	861.96725	2101.22	248.192872	

	Statistics
0	counterfactual_post_grading_price_mean
1	counterfactual_post_grading_price_mean
2	counterfactual_post_grading_price_mean

[94]:
```
price_statistics_df = pd.concat(price_statistics_list, ignore_index=True)
```

[95]:
```
price_statistics_df['trans_type'].replace({'lr':'Ramp Transaction',
                                           'mill':'Mill Transaction',
                                           'All':'All Transaction'
                                          }, inplace = True)
```

[96]:
```
price_statistics_df
```

[96]:

	trans_type	Mean	Median	Min	Max	\
0	lr//	1525.499627	1560.000000	780.00000	2040.00	
1	Mill Transaction	1497.622339	1470.000000	810.00000	2200.00	
2	All Transaction	1504.582584	1500.000000	780.00000	2200.00	
3	Ramp Transaction	1720.014573	1800.000000	1000.00000	2050.00	
4	Mill Transaction	1599.826647	1630.000000	905.00000	2200.00	
5	All Transaction	1647.721900	1700.000000	905.00000	2200.00	
6	Ramp Transaction	1653.256798	1713.617500	940.10000	1989.90	
7	Mill Transaction	1525.058727	1548.337000	861.96725	2101.22	
8	All Transaction	1575.628505	1623.248000	861.96725	2101.22	
9	Ramp Transaction	1700.512534	1760.000000	1030.00000	2065.00	
10	Mill Transaction	1597.161193	1630.000000	905.00000	2200.00	
11	All Transaction	1641.830298	1700.000000	905.00000	2200.00	
12	Ramp Transaction	1626.772090	1682.639000	973.59000	1969.50	

```
13  Mill Transaction  1521.894932  1547.397250   861.96725  2101.22
14   All Transaction  1567.034368  1619.661312   861.96725  2101.22

       Std Dev                                Statistics
0   287.834274                                     Price
1   280.336062                                     Price
2   282.475782                                     Price
3   208.536482                                     Price
4   285.212637                                     Price
5   264.018511                                     Price
6   195.429099                       Post Grading Price
7   274.451307                       Post Grading Price
8   254.139883                       Post Grading Price
9   206.736687                 counterfactual_price_mean
10  280.326405                 counterfactual_price_mean
11  256.303593                 counterfactual_price_mean
12  196.474156  counterfactual_post_grading_price_mean
13  272.542686  counterfactual_post_grading_price_mean
14  248.192872  counterfactual_post_grading_price_mean
```

[97]:
```
price_statistics_df['latex'] = (
                                price_statistics_df['Statistics'] +" & " +
                                price_statistics_df['trans_type'] + " & " +
                                price_statistics_df['Mean'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                price_statistics_df['Median'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) + " & " +
                                price_statistics_df['Min'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                price_statistics_df['Max'].map(lambda x: '{0:.
↪3f}'.format(x)).astype(str) + " & " +
                                price_statistics_df['Std Dev'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) +
                                "\\" + "\\"
                               )
```

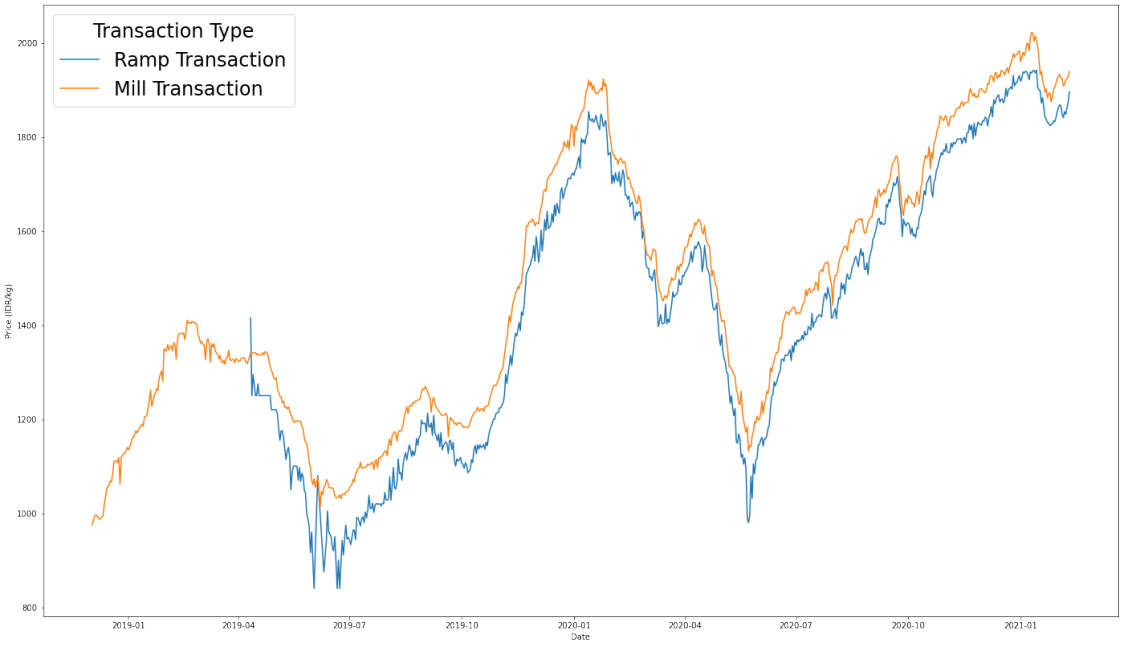
[98]:
```
price_statistics_df.to_excel(f"{output_path}" + "price_statistics_df.xlsx")
```

[99]:
```
counterfactual_price['Transaction Type']  = counterfactual_price['trans_type']
```

[100]:
```
counterfactual_price['Transaction Type'].replace({'lr//':'Ramp Transaction',␣
↪'mill' : 'Mill Transaction'}, inplace=True)
```

[101]:
```
fig, ax = plt.subplots(figsize=(24,14))

plt.rcParams.update({'font.size': 24})
```

```
sns.lineplot(x='date', y='price_counterfactual_table',data=counterfactual_price,
             hue='Transaction Type', ax = ax, ci=None)

plt.xlabel('Date')
plt.ylabel('Price (IDR/kg)')
plt.savefig(f"{fig_output_path}" +"mill_ramp_price.png")
```



3.4 Distance statistics

[102]:
```
mask = users_receipt['new_buyer'] == True

by_village_statistics = pd.pivot_table(users_receipt,
                                       index =␣
↪['counterfactual_buyer_in_the_same_village',
                                             'buyer_in_the_same_village'],
                                       values =␣
↪'post_grading_price_difference_%',
                                       aggfunc= [np.nanmean,lambda x: stats.
↪sem(x.dropna()),'count',
                                                 np.nanmin,np.nanmedian,np.
↪nanmax],
                                       margins=True
                                      )
```

```
by_village_statistics.columns = [column[0] +' Post Grading Price Difference %' ␣
↪for
                                 column in by_village_statistics.columns]

by_village_statistics.rename(columns= {'<lambda> Post Grading Price Difference␣
↪%':
                                       'SEM Post Grading Price Difference %'
                                      },
                             inplace = True
                            )

by_village_statistics
```

[102]:

```
                                                                    nanmean Post
Grading Price Difference %  \
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                    False
0.537796
                                         True
-0.488938
True                                     False
-0.260801
                                         True
0.024862
All
0.506659

                                                                     SEM Post
Grading Price Difference %  \
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                    False
0.035564
                                         True
0.409628
True                                     False
0.692511
                                         True
0.021457
All
0.034472

                                                                    count Post
Grading Price Difference %  \
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                    False
5054
                                         True
```

```
41
True                                    False
42
                                        True
175
All
5312

                                                                        nanmin Post
Grading Price Difference %  \
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                   False
-17.939268
                                        True
-6.344411
True                                    False
-14.526073
                                        True
-1.562500
All
-17.939268

                                                                        nanmedian
Post Grading Price Difference %  \
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                   False
0.000000
                                        True
-0.775987
True                                    False
-0.241997
                                        True
0.000000
All
0.000000

                                                                        nanmax Post
Grading Price Difference %
counterfactual_buyer_in_the_same_village buyer_in_the_same_village
False                                   False
30.251413
                                        True
5.039267
True                                    False
7.154981
                                        True
2.110174
```

```
All
30.251413
```

3.5 Write-up statistics

[103]:
```
users_receipt.columns
```

[103]:
```
Index(['id', 'trans_type', 'mill_id', 'lr_id', 'do_id', 'lrm_id', 'user_id',
       'weight_receipt', 'price_receipt', 'status',
       …
       'total_group_membership', 'nighttime_light', 'nighttime_light_density',
       'platform_age', 'price_change_class', 'delta_best_price_counterfactual',
       'delta_best_price_viewed_counterfactual', 'price_treatment_effect',
       'post_grading_price_treatment_effect', 'binary_role'],
      dtype='object', length=286)
```

[104]:
```
users_receipt['price_source'].value_counts()
```

[104]:
```
Transactions_sql              3327
price fruit sale              2194
Manual Imputation             1213
Counterfactual price table     739
special_price_daily            276
Name: price_source, dtype: int64
```

4 Visualization

[105]:
```
def plot_min_max_band_marker(data, x, y):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    fig, ax = plt.subplots(figsize=(10,7))

    plt.rcParams.update({'font.size': 12})

    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    plt.fill_between(tmp_df['index'], tmp_df['max'],
                     tmp_df['min'],
                    alpha=0.7, color='lightblue')
```

```
    scatter = ax.scatter(tmp_df['index'], tmp_df['mean'], color='black',
                         s = np.array(tmp_df['count'])
                         )

    #tmp_df.plot.scatter(['index'],['mean'],
    #                    s = tmp_df['count'],
    #                    color='black', ax = ax)

    # produce a legend with the unique colors from the scatter
    #legend1 = ax.legend(*scatter.legend_elements(),
    #                    loc="lower left", title="Classes")
    #ax.add_artist(legend1)

    # produce a legend with a cross section of sizes from the scatter
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
    legend2 = ax.legend(handles, labels, loc="lower right", title="Data Points")

    plt.axhline(y=0, linewidth=1, linestyle='dashed', color = 'grey')

    plt.xlabel(x)

    plt.xlim(0,234)

    #plt.xlim(20,None)

    plt.ylabel(y)
```

[106]:
```
def plot_new_buyer(data, x, y):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    fig, ax = plt.subplots(figsize=(20,14))

    plt.rcParams.update({'font.size': 18})

    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    #plt.fill_between(tmp_df['index'], tmp_df['max'],
    #                 tmp_df['min'],
    #                alpha=0.7, color='lightblue')

    scatter = ax.scatter(tmp_df['index'], tmp_df['mean'], color='black',
```

```
                        s = np.array(tmp_df['count'])
                        )

    #tmp_df.plot.scatter(['index'],['mean'],
    #                    s = tmp_df['count'],
    #                    color='black', ax = ax)

    # produce a legend with the unique colors from the scatter
    #legend1 = ax.legend(*scatter.legend_elements(),
    #                    loc="lower left", title="Classes")
    #ax.add_artist(legend1)

    # produce a legend with a cross section of sizes from the scatter
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
    legend2 = ax.legend(handles, labels, loc="lower right", title="Data Points")

    plt.xlabel('Upload Order')

    #plt.xlim(0,235)

    plt.ylim(0,1)

    plt.ylabel('Fraction New Buyer')
```

[107]:
```
def get_upload_order_bin(row):
    #print(row)
    upload_order = users_receipt['upload_order'].loc[row]
    #print(upload_order)
    output = np.nan

    if upload_order >90:
        output = '>90'
    else:
        output = str(math.ceil(upload_order/10)*10)

    return output
```

[108]:
```
def plot_new_buyer_cumulative(data, x, y):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    fig, ax = plt.subplots(figsize=(10,7))
```

```
    plt.rcParams.update({'font.size': 12})

    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    #plt.fill_between(tmp_df['index'], tmp_df['max'],
    #                 tmp_df['min'],
    #                alpha=0.7, color='lightblue')

    scatter = ax.scatter(tmp_df['index'], tmp_df['mean'], color='black',
                         s = np.array(tmp_df['count'])
                         )

    #tmp_df.plot.scatter(['index'],['mean'],
    #                    s = tmp_df['count'],
    #                    color='black', ax = ax)

    # produce a legend with the unique colors from the scatter
    #legend1 = ax.legend(*scatter.legend_elements(),
    #                    loc="lower left", title="Classes")
    #ax.add_artist(legend1)

    # produce a legend with a cross section of sizes from the scatter
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
    legend2 = ax.legend(handles, labels, loc="lower right", title="Data Points")

    plt.xlabel(x)

    plt.xlim(0,235)

    plt.ylim(0,1)

    plt.ylabel('Fraction New Buyer')
```

5 Regression analysis

5.1 Regression analysis with no covariate

[109]:
```
def create_regression_table(data,observed, counterfactual):
    cols = ['receipt_id','user_id',observed, counterfactual]

    regression_table = pd.melt(data[cols].dropna(),
                               id_vars=['receipt_id','user_id'],
                               var_name='variable',
                               value_name=observed
                              )
```

```
    regression_table['treatment_dummy']=␣
↪1*(regression_table['variable']==observed)

    if (regression_table['receipt_id'].value_counts()==2).mean()!=1 :
        output = False
    else :
        output = regression_table
    output = regression_table

    return output
```

[110]:
```
test = create_regression_table(users_receipt,'price_imputed_final',␣
↪'counterfactual_price_mean')
```

[111]:
```
def create_regression_table_size_larger_2(data,observed, counterfactual):
    cols = ['receipt_id','user_id',observed, counterfactual]
    print(users_receipt[cols].dropna().shape[0]*2)

    regression_table = pd.melt(data[cols].dropna(),
                               id_vars=['receipt_id','user_id'],
                               var_name='variable',
                               value_name=observed
                              )

    ## Filtering users with at least 2 data points
    data_point = pd.pivot_table(regression_table, index = 'user_id',
                                values='receipt_id', aggfunc='count')
    data_point.reset_index(inplace=True)

    #print(data_point.columns)

    mask = data_point['receipt_id']>2

    user_id_list = list(data_point['user_id'][mask])

    ## Create regression table
    mask_2 = regression_table['user_id'].isin(user_id_list)

    regression_table = regression_table[mask_2]

    regression_table['treatment_dummy']=␣
↪1*(regression_table['variable']==observed)

    if (regression_table['receipt_id'].value_counts()==2).mean()!=1 :
        output = False
    else :
```

```
        output = regression_table

    return regression_table
```

[112]:
```
def create_regression_table_size_1_old_buyer(data,observed, counterfactual):
    cols = ['receipt_id','user_id',observed, counterfactual]
    print(users_receipt[cols].dropna().shape[0]*2)

    regression_table = pd.melt(data[cols].dropna(),
                               id_vars=['receipt_id','user_id'],
                               var_name='variable',
                               value_name=observed
                              )

    ## Filtering users with at least 2 data points
    data_point = pd.pivot_table(regression_table, index = 'user_id',
                                values='receipt_id', aggfunc='count')
    data_point.reset_index(inplace=True)

    #print(data_point.columns)

    mask = users['old_buyers_per_user'] ==1

    user_id_list = list(users['user_id'][mask])

    ## Create regression table
    mask_2 = regression_table['user_id'].isin(user_id_list)

    regression_table = regression_table[mask_2]

    regression_table['treatment_dummy']=␣
↪1*(regression_table['variable']==observed)

    if (regression_table['receipt_id'].value_counts()==2).mean()!=1 :
        output = False
    else :
        output = regression_table

    return regression_table
```

[113]:
```
def generate_regression_result(data,table_func,observed,␣
↪counterfactual,output_label):
    output = {}

    data_2 = table_func(data,observed, counterfactual).dropna()
    dependent_variable = data_2[data_2.columns[3]]
    regressors = sm.add_constant(data_2[data_2.columns[4]].astype(float))
```

```
    mod = sm.OLS(dependent_variable.astype(float), regressors.astype(float))
    res = mod.fit()

    output['number_cluster'] = len(data_2['user_id'].unique())
    output['number_transaction'] = len(data_2)/2
    output['effect'] = res.params['treatment_dummy']
    output['p_values'] = res.pvalues['treatment_dummy']

    output = pd.DataFrame.from_dict(output,orient='index')

    output.columns = [output_label]

    output = output.transpose()

    return output
```

[114]:
```
def generate_cluster_regression_result(data,table_func,observed,␣
↪counterfactual,output_label):
    output = {}

    data_2 = table_func(data,observed, counterfactual).dropna()
    #print(len(data_2['user_id'].unique()))
    dependent_variable = data_2[data_2.columns[3]]
    regressors = sm.add_constant(data_2[data_2.columns[4]].astype(float))
    mod = sm.OLS(dependent_variable.astype(float), regressors.astype(float))
    res = mod.fit(cov_type='cluster',
              cov_kwds={'groups': data_2['user_id']},
              use_correction = True,
                  missing='drop'
             )
    #print(res.params)
    output['number_cluster'] = len(data_2['user_id'].unique())
    output['number_transaction'] = len(data_2)/2
    output['effect'] = res.params['treatment_dummy']
    output['std_err_treatment'] = res.bse['treatment_dummy']
    output['p_values_treatment'] = res.pvalues['treatment_dummy']
    output['intercept'] = res.params['const']
    output['p_values_intercept'] = res.pvalues['const']

    output = pd.DataFrame.from_dict(output,orient='index')

    output.columns = [output_label]

    output = output.transpose()

    return output
```

[115]:
```
generate_cluster_regression_result(users_receipt,
                                   ␣
 ↪create_regression_table,'price_imputed_final', 'counterfactual_price_mean',
                                    'output_label')
```

[115]:

	number_cluster	number_transaction	effect	std_err_treatment	p_values_treatment	intercept	p_values_intercept
output_label	391.0	5673.0	5.211565	1.935827	0.007099	1643.128114	0.0

[116]:
```
counterfactual_obs_output_list_mean = [['price_imputed_final',␣
 ↪'counterfactual_price_mean','Price'],
                                       ␣
 ↪['best_price','counterfactual_price_max','Best Price'],
                                       ␣
 ↪['best_price_viewed','counterfactual_price_max','Best Price Viewed'],
                                       ␣
 ↪['post_grading_price','counterfactual_post_grading_price_mean','Post Grading␣
 ↪Price'],
                                       ␣
 ↪['best_post_grading_price','counterfactual_post_grading_price_max','Best␣
 ↪Post Grading Price'],
                                       ␣
 ↪['best_post_grading_price_viewed','counterfactual_post_grading_price_max','Best␣
 ↪Post Grading Price Viewed'],
                                       ␣
 ↪['grade_imputed_final','counterfactual_grade_mean','Grade'],
                                       ␣
 ↪['distance','counterfactual_distance_mean','Distance'],
                                        #["Post Grading Price per␣
 ↪Distance",'Counterfactual Post Grading Price per Distance','Post Grading␣
 ↪Price per Distance'],
                                       ␣
 ↪['revenue','counterfactual_revenue_mean','Revenue']
                                      ]
```

[117]:
```
counterfactual_obs_output_list_max = [['price_imputed_final',␣
 ↪'counterfactual_price_max','Price'],
                                       ␣
 ↪['best_price','counterfactual_price_max','Best Price'],
                                       ␣
 ↪['best_price_viewed','counterfactual_price_max','Best Price Viewed'],
                                       ␣
 ↪['post_grading_price','counterfactual_post_grading_price_max','Post Grading␣
 ↪Price'],
```

```
                                        ␣
↪['best_post_grading_price','counterfactual_post_grading_price_max','Best␣
↪Post Grading Price'],
                                        ␣
↪['best_post_grading_price_viewed','counterfactual_post_grading_price_max','Best␣
↪Post Grading Price Viewed'],
                                        ␣
↪#['grade_imputed_final','counterfactual_grade_max','Grade'],
                                        ␣
↪#['distance','counterfactual_distance','Distance'],
                                            #["Post Grading Price per␣
↪Distance",'Counterfactual Post Grading Price per Distance','Post Grading␣
↪Price per Distance'],
                                        ␣
↪['revenue','counterfactual_revenue_max','Revenue']
                                       ]
```

[118]:
```
counterfactual_obs_output_list_robust = [['price_imputed_final',␣
↪'counterfactual_price_special_group_robust','Price'],
                                        ␣
↪['best_price','counterfactual_price_special_group_robust_max','Best Price'],
                                        ␣
↪['best_price_viewed','counterfactual_price_special_group_robust_max','Best␣
↪Price Viewed'],
                                        ␣
↪['post_grading_price','counterfactual_post_grading_price_special_group_robust','Post␣
↪Grading Price'],
                                        ␣
↪['best_post_grading_price','counterfactual_post_grading_price_special_group_robust_max','Be
↪Post Grading Price'],
                                        ␣
↪['best_post_grading_price_viewed','counterfactual_post_grading_price_special_group_robust_m
↪Post Grading Price Viewed'],
                                        ␣
↪#['grade_imputed_final','counterfactual_grade_max','Grade'],
                                        ␣
↪#['distance','counterfactual_distance','Distance'],
                                            #["Post Grading Price per␣
↪Distance",'Counterfactual Post Grading Price per Distance','Post Grading␣
↪Price per Distance'],
                                        ␣
↪['revenue','counterfactual_revenue_special_group_robust','Revenue']
                                       ]
```

[119]:
```
def get_regression_result(regression_func, table_func,␣
↪data,subset_name,counterfactual_obs_output_list):
    regression_result = {}

    for i in range(len(counterfactual_obs_output_list)):
        #print(i)
        #print(counterfactual_obs_output_list[i][0])
        output = regression_func(data,
                                 table_func,
                                 counterfactual_obs_output_list[i][0],
                                 counterfactual_obs_output_list[i][1],
                                 counterfactual_obs_output_list[i][2]
                                 )
        regression_result[counterfactual_obs_output_list[i][2]] = output

    regression_result_df = regression_result['Price']

    for key in regression_result.keys():
        #print(regression_result[key])
        regression_result_df = pd.
↪concat([regression_result_df,regression_result[key]])
        regression_result_df.drop_duplicates(inplace=True)

    regression_result_df['subset'] = subset_name
    regression_result_df.reset_index(inplace=True)
    regression_result_df.set_index(['subset','index'], inplace= True)
    regression_result_df['effect_percent'] = regression_result_df['effect']/
↪regression_result_df['intercept']*100

    return regression_result_df
```

[120]:
```
regression_result_df = get_regression_result(generate_cluster_regression_result,
                                             create_regression_table,
                                             users_receipt,
                                             'All Data Mean',
                                            counterfactual_obs_output_list_mean
                                           )
```

[121]:
```
regression_result_df = pd.concat([regression_result_df,
                                  ␣
↪get_regression_result(generate_cluster_regression_result,
                                                        create_regression_table,
                                                        users_receipt,
                                                        'All Data Max',
                                                       ␣
↪counterfactual_obs_output_list_max
                                                        )
```

```
                                    ],
                                    #ignore_index= True,
                                   )
```

[122]:
```
regression_result_df = pd.concat([regression_result_df,
                                  ␣
 ↪get_regression_result(generate_cluster_regression_result,
                                                         create_regression_table,
                                                         users_receipt,
                                                         'All Data Special Group␣
 ↪Robust',
                                                        ␣
 ↪counterfactual_obs_output_list_robust
                                                         )
                                    ],
                                    #ignore_index= True,
                                   )
```

[123]:
```
users_receipt.shape
```

[123]:
```
(7933, 286)
```

[124]:
```
regression_result_df.columns
```

[124]:
```
Index(['number_cluster', 'number_transaction', 'effect', 'std_err_treatment',
       'p_values_treatment', 'intercept', 'p_values_intercept',
       'effect_percent'],
      dtype='object')
```

[125]:
```
regression_result_df['effect_percent'] = regression_result_df['effect']/
 ↪regression_result_df['intercept']*100

regression_result_df.reset_index(inplace= True)
```

[126]:
```
regression_result_df['latex'] = ("\\" + "\\" +
                                 regression_result_df['index'] +" & " +
                                 regression_result_df['number_cluster'].
 ↪astype(str).replace('\.0', '', regex=True) +" & " +
                                 regression_result_df['number_transaction'].
 ↪astype(str).replace('\.0', '', regex=True) +" & " +
                                 regression_result_df['effect'].map(lambda x:␣
 ↪'{0:.3f}'.format(x)).astype(str) + " & " +
                                 regression_result_df['std_err_treatment'].
 ↪map(lambda x: '{0:.3f}'.format(x)).astype(str) + " & " +
                                 regression_result_df['p_values_treatment'].
 ↪map(lambda x: '{0:.3f}'.format(x)).astype(str) + " & " +
```

```
                                    regression_result_df['effect_percent'].
↪map(lambda x: '{0:.3f}'.format(x)).astype(str) +
                                    "\\" + "\\\hline "
                                   )
```

[127]:
```
regression_result_df.to_excel(f"{output_path}" + "regression_result_df.xlsx")
```

[128]:
```
users_receipt.shape
```

[128]:
```
(7933, 286)
```

5.2 Rolling average upload order

[129]:
```
users_receipt['rolling_reg_subset'] = False

mask = ((users_receipt['post_grading_price'].notna()) &
        (users_receipt['counterfactual_post_grading_price_mean'].notna()) &
        (users_receipt['best_post_grading_price'].notna()) &
        (users_receipt['best_post_grading_price_viewed'].notna())
       )

users_receipt['rolling_reg_subset'][mask] = True

users_receipt['rolling_reg_subset'].sum()
```

[129]:
```
5308
```

[130]:
```
def get_rolling_regression(data_used,max_upload_order,window_size):
    regression_result_rolling =␣
↪get_regression_result(generate_cluster_regression_result,
                                                create_regression_table,
                                                data_used,
                                                'All Data Mean',
                                               counterfactual_obs_output_list_mean
                                              )
    for i in range(window_size, max_upload_order):
        #print(i)
        mask = ((data_used['upload_order']>i-window_size) &
                (data_used['upload_order']<=i) &
                 data_used['rolling_reg_subset']
               )

        data = data_used[mask]

        regression_result =␣
↪get_regression_result(generate_cluster_regression_result,
```

```
                                            create_regression_table,
                                            data,
                                            i,
                                           counterfactual_obs_output_list_mean
                                          )

        regression_result_rolling = pd.concat([regression_result_rolling,␣
↪regression_result])

    regression_result_rolling.reset_index(inplace=True)
    mask = (#(regression_result_rolling['index']=='post_grading_price') &
        (regression_result_rolling['subset']!='All Data Mean')
       )

    regression_result_rolling = regression_result_rolling[mask]

    regression_result_rolling['lower'] = regression_result_rolling['effect'] -␣
↪1.96*regression_result_rolling['std_err_treatment']

    regression_result_rolling['upper'] = regression_result_rolling['effect'] +␣
↪1.96*regression_result_rolling['std_err_treatment']

    regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

    regression_result_rolling['subset'] = regression_result_rolling['subset'].
↪astype(int)

    regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

    regression_result_rolling['lower_%'] = regression_result_rolling['lower']/
↪regression_result_rolling['intercept']*100

    regression_result_rolling['upper_%'] = regression_result_rolling['upper']/
↪regression_result_rolling['intercept']*100

    return regression_result_rolling
```

[131]:
```
regression_result_rolling = get_rolling_regression(users_receipt,339,100)
```

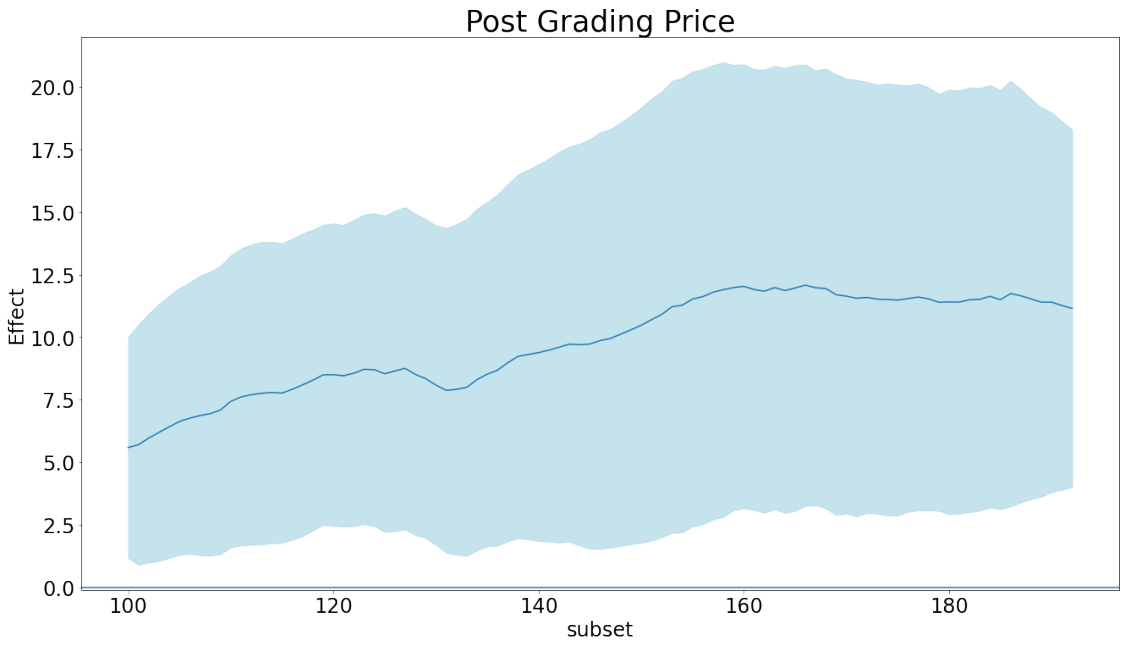
[132]:
```
regression_result_rolling.to_excel(f"{output_path}" +␣
↪"regression_result_rolling.xlsx")
```

[133]:
```
def plot_reg_rolling(data, variable):
    mask = (data['index']==variable)
```

```
    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(22, 12))
    plt.rcParams.update({'font.size': 35})

    sns.lineplot(x='subset', y='effect', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper'],
                     tmp_df['lower'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable,fontsize=35)
    plt.ylabel('Effect')
    #plt.xlim(None, 140)
    #plt.ylim(None, 160)
```

[134]:
```
def get_slope_rolling_reg(data, variable):
    mask = (data['index']==variable)
    Y = data['effect'][mask]
    X = sm.add_constant(data['subset'][mask]-99)

    mod = sm.OLS(Y,X)

    results = mod.fit()
    print(results.summary())
```

[135]:
```
mask = regression_result_rolling['subset'] <193

plot_reg_rolling(regression_result_rolling[mask], 'Post Grading Price')
plt.savefig(f"{fig_output_path}" +"post_grading_price_rolling_regression.png")
```



[136]:
```
mask = regression_result_rolling['subset'] <193

get_slope_rolling_reg(regression_result_rolling[mask], 'Post Grading Price')
```

OLS Regression Results

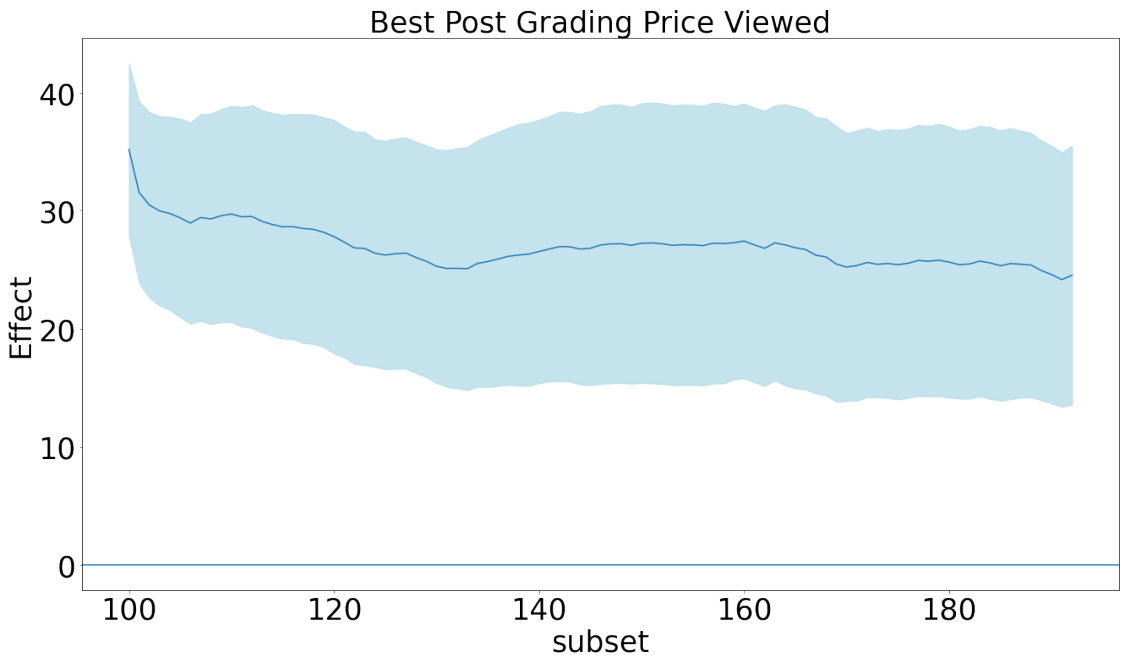
Dep. Variable:	effect	R-squared:	0.864
Model:	OLS	Adj. R-squared:	0.863
Method:	Least Squares	F-statistic:	579.7
Date:	Sun, 29 Aug 2021	Prob (F-statistic):	3.04e-41
Time:	20:36:29	Log-Likelihood:	-97.797
No. Observations:	93	AIC:	199.6
Df Residuals:	91	BIC:	204.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.7089	0.146	45.832	0.000	6.418	7.000
subset	0.0651	0.003	24.076	0.000	0.060	0.070

Omnibus:	1.176	Durbin-Watson:	0.039
Prob(Omnibus):	0.555	Jarque-Bera (JB):	1.049
Skew:	0.049	Prob(JB):	0.592
Kurtosis:	2.489	Cond. No.	109.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[137]:
```
mask = regression_result_rolling['subset'] <193
plot_reg_rolling(regression_result_rolling[mask], 'Best Post Grading Price␣
↪Viewed')
plt.savefig(f"{fig_output_path}"␣
↪+"best_post_grading_price_viewed_rolling_regression.png")
```



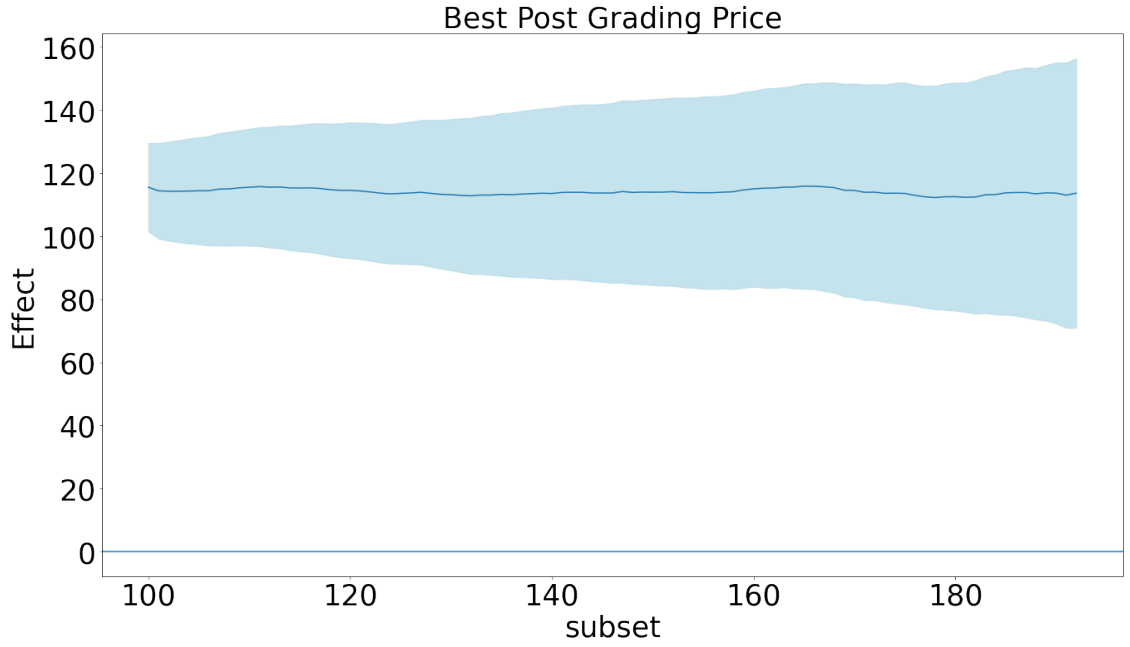
[138]:
```
mask = regression_result_rolling['subset'] <193

get_slope_rolling_reg(regression_result_rolling[mask], 'Best Post Grading Price␣
↪Viewed')
```

OLS Regression Results

Dep. Variable:	effect	R-squared:	0.552
Model:	OLS	Adj. R-squared:	0.547
Method:	Least Squares	F-statistic:	112.0
Date:	Sun, 29 Aug 2021	Prob (F-statistic):	1.57e-17
Time:	20:36:30	Log-Likelihood:	-145.66
No. Observations:	93	AIC:	295.3
Df Residuals:	91	BIC:	300.4
Df Model:	1		
Covariance Type:	nonrobust		

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         29.2058      0.245    119.256      0.000      28.719      29.692
subset        -0.0479      0.005    -10.582      0.000      -0.057      -0.039
==============================================================================
Omnibus:                       34.318   Durbin-Watson:                   0.156
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              191.642
Skew:                           0.930   Prob(JB):                     2.43e-42
Kurtosis:                       9.782   Cond. No.                         109.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[139]:
```
mask = regression_result_rolling['subset'] <193
plot_reg_rolling(regression_result_rolling[mask], 'Best Post Grading Price')
plt.savefig(f"{fig_output_path}" +"best_post_grading_price_rolling_regression.
 ↪png")
```



[140]:
```
mask = regression_result_rolling['subset'] <193

get_slope_rolling_reg(regression_result_rolling[mask], 'Best Post Grading␣
 ↪Price')
```

OLS Regression Results

```
==============================================================================
Dep. Variable:                  effect   R-squared:                       0.147
Model:                             OLS   Adj. R-squared:                  0.137
Method:                  Least Squares   F-statistic:                     15.62
Date:                 Sun, 29 Aug 2021   Prob (F-statistic):           0.000153
Time:                         20:36:30   Log-Likelihood:                -114.51
No. Observations:                   93   AIC:                             233.0
Df Residuals:                       91   BIC:                             238.1
Df Model:                            1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        114.6192      0.175    654.260      0.000     114.271     114.967
subset        -0.0128      0.003     -3.952      0.000      -0.019      -0.006
==============================================================================
Omnibus:                        4.209   Durbin-Watson:                   0.130
Prob(Omnibus):                  0.122   Jarque-Bera (JB):                4.192
Skew:                           0.510   Prob(JB):                        0.123
Kurtosis:                       2.793   Cond. No.                         109.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[141]:
```
def get_rolling_regression_max(data_used,max_upload_order,window_size):
    regression_result_rolling =␣
↪get_regression_result(generate_cluster_regression_result,
                                                create_regression_table,
                                                data_used,
                                                'All Data Max',
                                               counterfactual_obs_output_list_max
                                              )
    for i in range(window_size, max_upload_order):
        #print(i)
        mask = ((data_used['upload_order']>i-window_size) &
                (data_used['upload_order']<=i) &
                 data_used['rolling_reg_subset']
               )

        data = data_used[mask]

        regression_result =␣
↪get_regression_result(generate_cluster_regression_result,
                                                create_regression_table,
                                                data,
```

```
                                            i,
                                            counterfactual_obs_output_list_max
                                           )

       regression_result_rolling = pd.concat([regression_result_rolling,␣
↪regression_result])

   regression_result_rolling.reset_index(inplace=True)
   mask = (#(regression_result_rolling['index']=='post_grading_price') &
       (regression_result_rolling['subset']!='All Data Max')
      )

   regression_result_rolling = regression_result_rolling[mask]

   regression_result_rolling['lower'] = regression_result_rolling['effect'] -␣
↪1.96*regression_result_rolling['std_err_treatment']

   regression_result_rolling['upper'] = regression_result_rolling['effect'] +␣
↪1.96*regression_result_rolling['std_err_treatment']

   regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

   regression_result_rolling['subset'] = regression_result_rolling['subset'].
↪astype(int)

   regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

   regression_result_rolling['lower_%'] = regression_result_rolling['lower']/
↪regression_result_rolling['intercept']*100

   regression_result_rolling['upper_%'] = regression_result_rolling['upper']/
↪regression_result_rolling['intercept']*100

   return regression_result_rolling
```

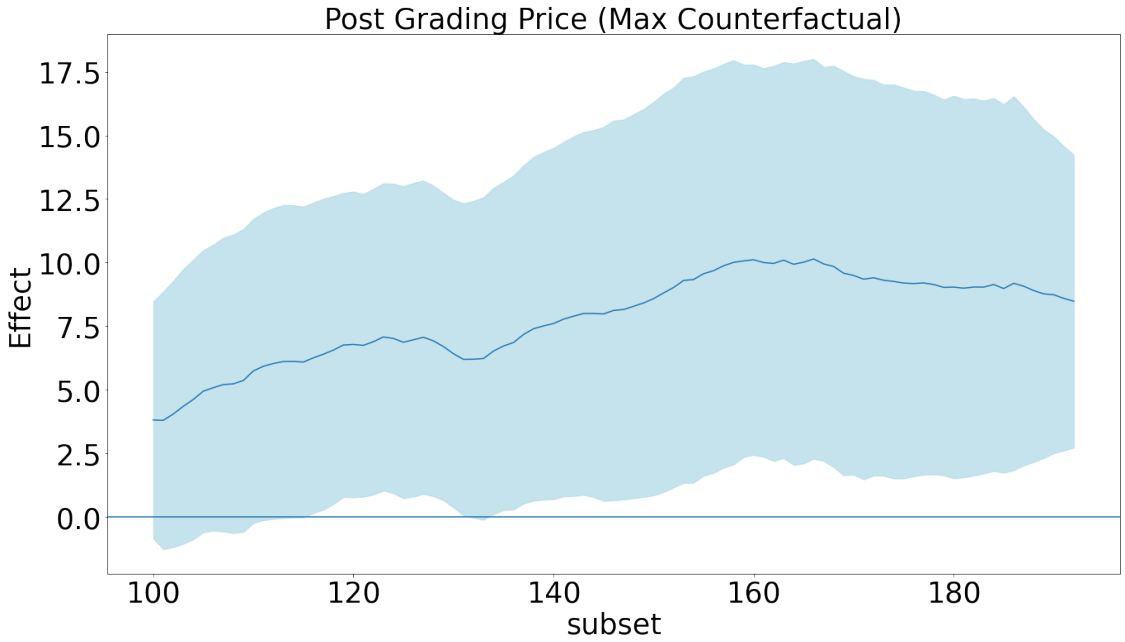
[142]:
```
regression_result_rolling_max =␣
↪get_rolling_regression_max(users_receipt,339,100)
```

[143]:
```
regression_result_rolling_max.to_excel(f"{output_path}" +␣
↪"regression_result_rolling_max.xlsx")
```

[144]:
```
def plot_reg_rolling_max(data, variable):
    mask = (data['index']==variable)
```

```
    tmp_df = data[mask].dropna().sort_values(by='subset')
    plt.rcParams.update({'font.size': 35})
    fig, ax = plt.subplots(figsize=(22, 12))

    sns.lineplot(x='subset', y='effect', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper'],
                     tmp_df['lower'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable + " (Max Counterfactual)",fontsize=35)
    plt.ylabel('Effect')
    #plt.xlim(None, 140)
    #plt.ylim(None, 160)
```

[145]:
```
mask = regression_result_rolling_max['subset'] <193

plot_reg_rolling_max(regression_result_rolling_max[mask], 'Post Grading Price')
plt.savefig(f"{fig_output_path}" +"post_grading_price_rolling_regression_max.
 ↪png")
```



[146]:
```
mask = regression_result_rolling_max['subset'] <193
get_slope_rolling_reg(regression_result_rolling_max[mask], 'Post Grading Price')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 effect   R-squared:                       0.769
Model:                            OLS   Adj. R-squared:                  0.767
Method:                 Least Squares   F-statistic:                     303.7
Date:                Sun, 29 Aug 2021   Prob (F-statistic):           9.63e-31
Time:                        20:37:18   Log-Likelihood:                -112.62
No. Observations:                  93   AIC:                             229.2
Df Residuals:                      91   BIC:                             234.3
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.2233      0.172     30.426      0.000       4.882       5.564
subset         0.0553      0.003     17.427      0.000       0.049       0.062
==============================================================================
Omnibus:                        1.254   Durbin-Watson:                   0.032
Prob(Omnibus):                  0.534   Jarque-Bera (JB):                1.116
Skew:                          -0.081   Prob(JB):                        0.572
Kurtosis:                       2.488   Cond. No.                         109.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[147]:
```
def get_rolling_regression_robust(data_used,max_upload_order,window_size):
    regression_result_rolling =␣
↪get_regression_result(generate_cluster_regression_result,
                                                create_regression_table,
                                                data_used,
                                                'All Data Robust',
                                               ␣
↪counterfactual_obs_output_list_robust
                                               )
    for i in range(window_size, max_upload_order):
        #print(i)
        mask = ((data_used['upload_order']>i-window_size) &
                (data_used['upload_order']<=i) &
                 data_used['rolling_reg_subset']
               )

        data = data_used[mask]

        regression_result =␣
↪get_regression_result(generate_cluster_regression_result,
```

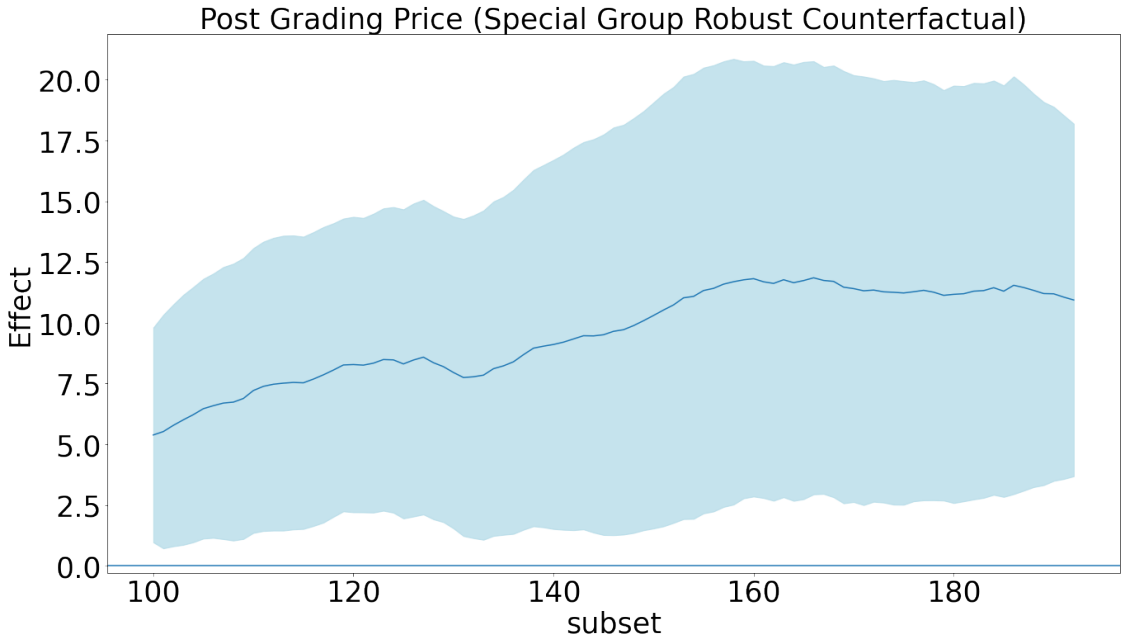
```
                                            create_regression_table,
                                            data,
                                            i,
                                        ␣
↪counterfactual_obs_output_list_robust
                                            )

      regression_result_rolling = pd.concat([regression_result_rolling,␣
↪regression_result])

   regression_result_rolling.reset_index(inplace=True)
   mask = (#(regression_result_rolling['index']=='post_grading_price') &
       (regression_result_rolling['subset']!='All Data Robust')
      )

   regression_result_rolling = regression_result_rolling[mask]

   regression_result_rolling['lower'] = regression_result_rolling['effect'] -␣
↪1.96*regression_result_rolling['std_err_treatment']

   regression_result_rolling['upper'] = regression_result_rolling['effect'] +␣
↪1.96*regression_result_rolling['std_err_treatment']

   regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

   regression_result_rolling['subset'] = regression_result_rolling['subset'].
↪astype(int)

   regression_result_rolling['effect_%'] = regression_result_rolling['effect']/
↪ regression_result_rolling['intercept']*100

   regression_result_rolling['lower_%'] = regression_result_rolling['lower']/
↪regression_result_rolling['intercept']*100

   regression_result_rolling['upper_%'] = regression_result_rolling['upper']/
↪regression_result_rolling['intercept']*100

   return regression_result_rolling
```

[148]:
```
regression_result_rolling_robust =␣
↪get_rolling_regression_robust(users_receipt,339,100)
```

[149]:
```
regression_result_rolling_robust.to_excel(f"{output_path}" +␣
↪"regression_result_rolling_robust.xlsx")
```

[150]:
```
def plot_reg_rolling_robust(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    plt.rcParams.update({'font.size': 35})
    fig, ax = plt.subplots(figsize=(22, 12))

    sns.lineplot(x='subset', y='effect', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper'],
                     tmp_df['lower'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable + " (Special Group Robust Counterfactual)",fontsize=35)
    plt.ylabel('Effect')
    #plt.xlim(None, 140)
    #plt.ylim(None, 160)
```

[151]:
```
mask = regression_result_rolling_robust['subset'] <193
plot_reg_rolling_robust(regression_result_rolling_robust[mask], 'Post Grading␣
↪Price')
plt.savefig(f"{fig_output_path}" +"post_grading_price_rolling_regression_robust.
↪png")
```



[152]:
```
mask = regression_result_rolling_robust['subset'] <193
get_slope_rolling_reg(regression_result_rolling_robust[mask], 'Post Grading␣
↪Price')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 effect   R-squared:                       0.865
Model:                            OLS   Adj. R-squared:                  0.864
Method:                 Least Squares   F-statistic:                     584.4
Date:                Sun, 29 Aug 2021   Prob (F-statistic):           2.20e-41
Time:                        20:38:04   Log-Likelihood:                -97.038
No. Observations:                  93   AIC:                             198.1
Df Residuals:                      91   BIC:                             203.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.5045      0.145     44.800      0.000       6.216       6.793
subset         0.0648      0.003     24.175      0.000       0.060       0.070
==============================================================================
Omnibus:                        1.210   Durbin-Watson:                   0.038
Prob(Omnibus):                  0.546   Jarque-Bera (JB):                1.135
Skew:                           0.117   Prob(JB):                        0.567
Kurtosis:                       2.512   Cond. No.                         109.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[153]:
```
mask = ((users_receipt['role'] =='Amprah Farmer') &
        (users_receipt['rolling_reg_subset']))

users_receipt['upload_order'][mask].max()
```

[153]: 129

[154]:
```
def plot_reg_rolling_obs(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(20, 8.27))

    sns.lineplot(x='subset', y='number_transaction', data=tmp_df)

    #sns.lineplot(x='subset', y='number_cluster', data=tmp_df)
```

```
    plt.axhline(y=50)

    plt.legend(['number_transaction'])

    plt.title(variable)
```

[155]:
```
def plot_reg_rolling_cluster(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(20, 8.27))

    #sns.lineplot(x='subset', y='number_transaction', data=tmp_df)

    sns.lineplot(x='subset', y='number_cluster', data=tmp_df, color='red')

    plt.axhline(y=20)

    plt.legend(['number_cluster'])

    plt.title(variable)
```

[156]:
```
def plot_reg_rolling_perc(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(20, 8.27))

    sns.lineplot(x='subset', y='effect_%', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper_%'],
                     tmp_df['lower_%'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable)
    plt.ylabel("Effect in %")
    #plt.ylim(None, 140)
```

[157]:
```
def plot_reg_rolling_perc_max(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(20, 8.27))
```

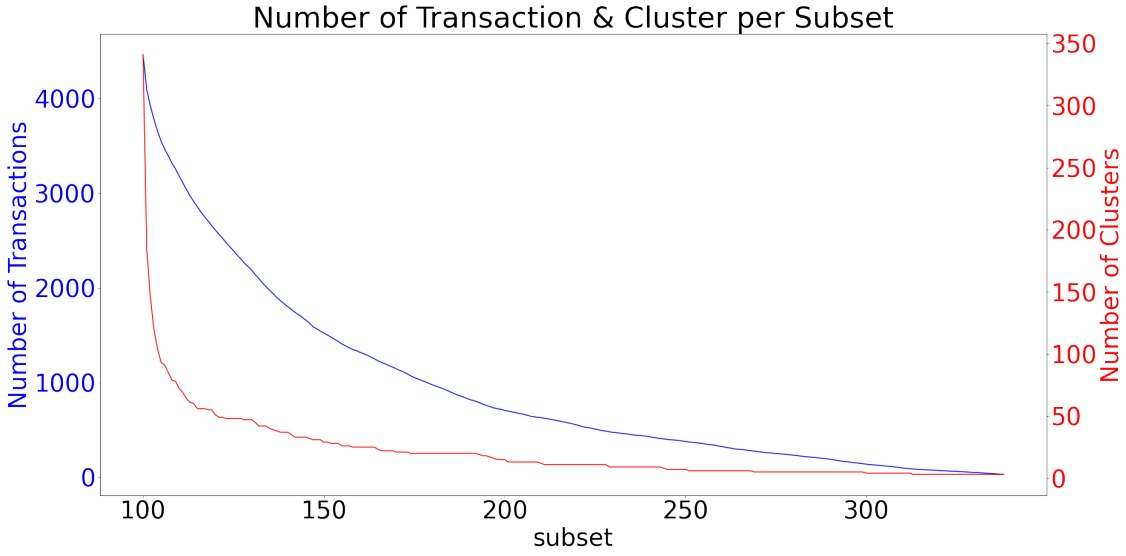
```
    sns.lineplot(x='subset', y='effect_%', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper_%'],
                     tmp_df['lower_%'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable + " (Max Counterfactual)")
    plt.ylabel("Effect in %")
    #plt.ylim(None, 140)
```

[158]:
```
def plot_reg_rolling_perc_robust(data, variable):
    mask = (data['index']==variable)

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax = plt.subplots(figsize=(20, 8.27))

    sns.lineplot(x='subset', y='effect_%', data=tmp_df)

    plt.fill_between(tmp_df['subset'],
                     tmp_df['upper_%'],
                     tmp_df['lower_%'],
                     alpha=0.7, color='lightblue'
                    )

    plt.axhline(y=0)
    plt.title(variable + " (Special Group Robust Counterfactual)")
    plt.ylabel("Effect in %")
    #plt.ylim(None, 140)
```

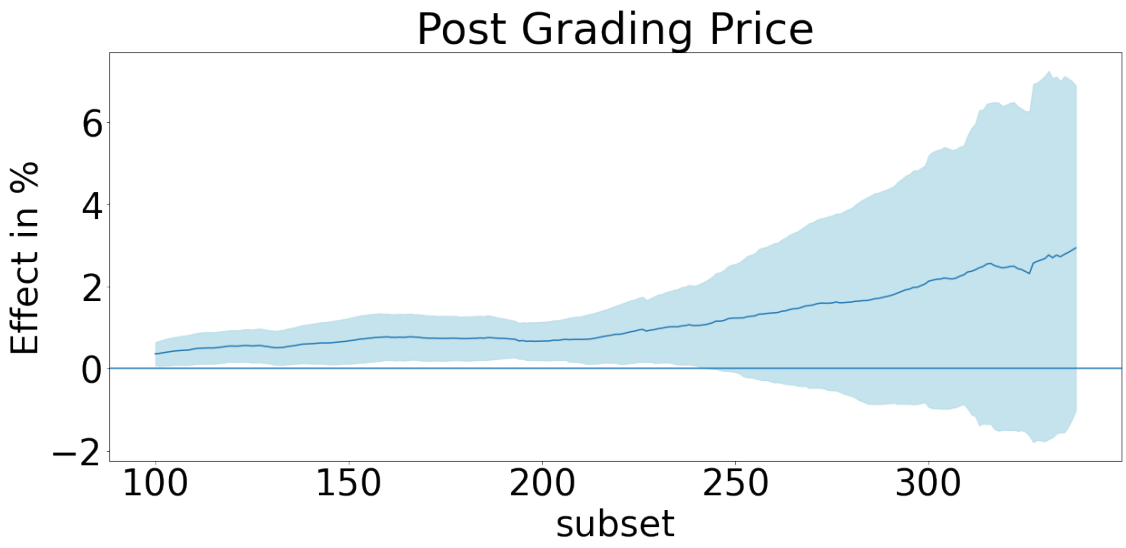
[159]:
```
def plot_reg_rolling_obs_cluster(data, variable):
    mask = (data['index']==variable)
    plt.rcParams.update({'font.size': 40})

    tmp_df = data[mask].dropna().sort_values(by='subset')
    fig, ax1 = plt.subplots(figsize=(28, 14))
    ax2 = ax1.twinx()

    #sns.lineplot(x='subset', y='number_transaction', data=tmp_df)

    ax1.plot(tmp_df['subset'], tmp_df['number_transaction'], color='blue')
    ax2.plot(tmp_df['subset'], tmp_df['number_cluster'], color='red')

    #ax1.legend(['number_transaction'])
```

```
    #ax2.legend(['number_cluster'])

    ax1.set_xlabel('subset')
    ax1.set_ylabel('Number of Transactions', color='blue')
    ax1.tick_params(axis='y', labelcolor='blue')
    ax2.set_ylabel('Number of Clusters', color='red')
    ax2.tick_params(axis='y', labelcolor='red')

    plt.title('Number of Transaction & Cluster per Subset')
```

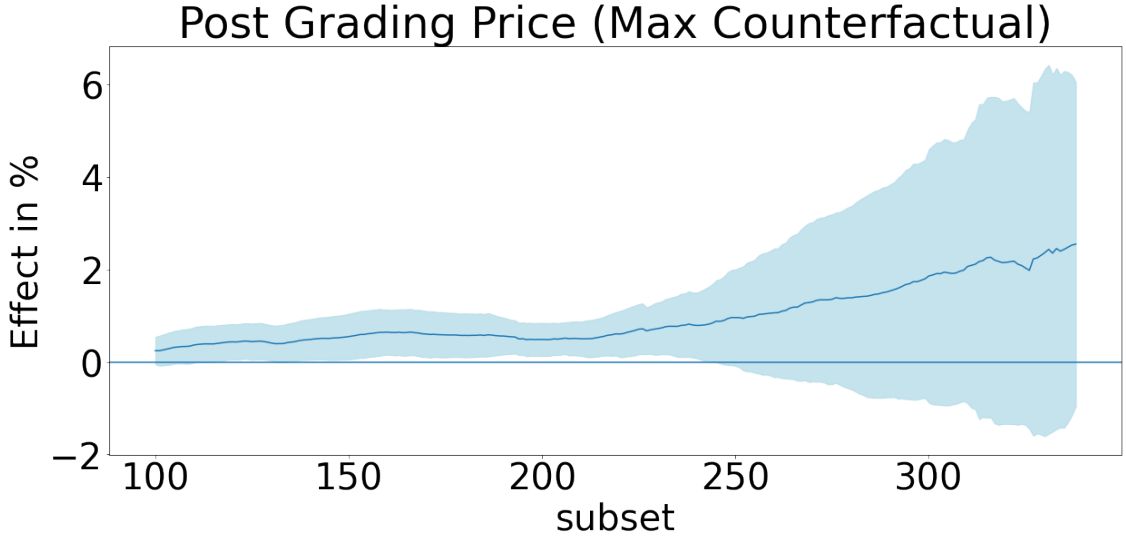
[160]:
```
plot_reg_rolling_obs_cluster(regression_result_rolling, 'Post Grading Price')
plt.savefig(f"{fig_output_path}" +"rolling_num_trx_cluster.png")
```



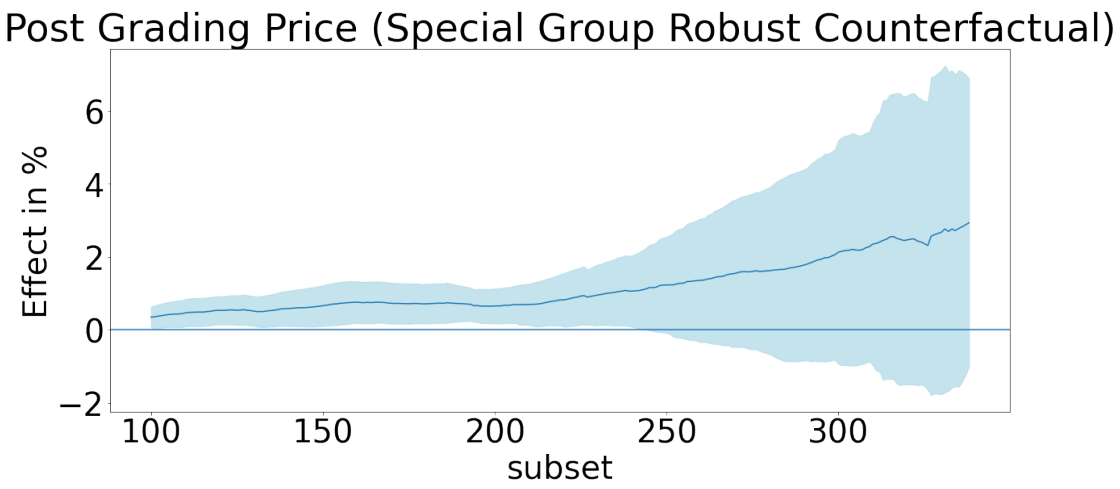
[161]:
```
plot_reg_rolling_perc(regression_result_rolling, 'Post Grading Price')
plt.savefig(f"{fig_output_path}" +"post_grading_price_rolling_regression_perc.
 ↪png")
```



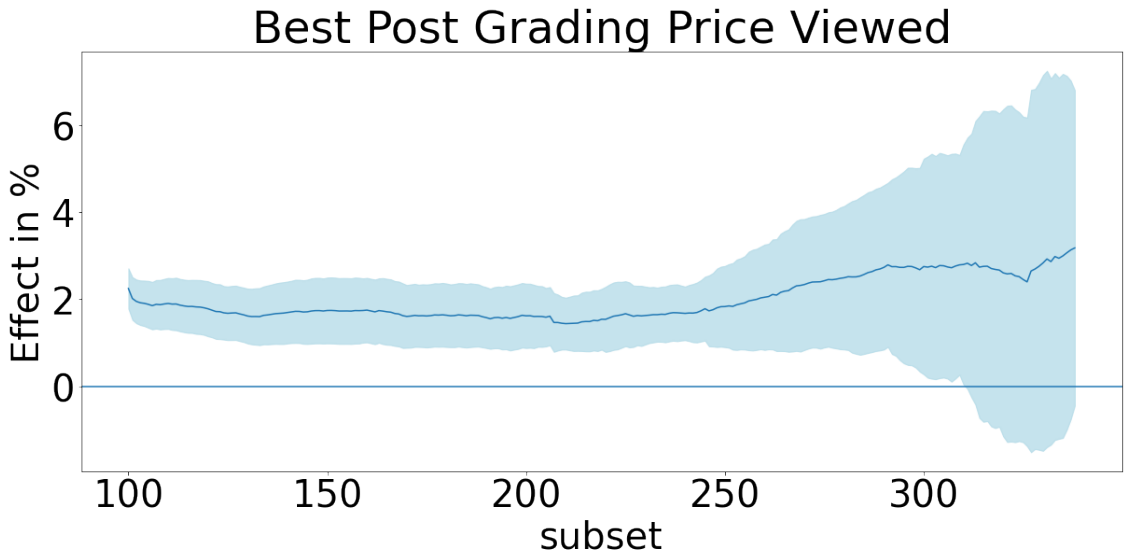
[162]:
```
plot_reg_rolling_perc_max(regression_result_rolling_max, 'Post Grading Price')
plt.savefig(f"{fig_output_path}"␣
 ↪+"post_grading_price_rolling_regression_max_perc.png")
```



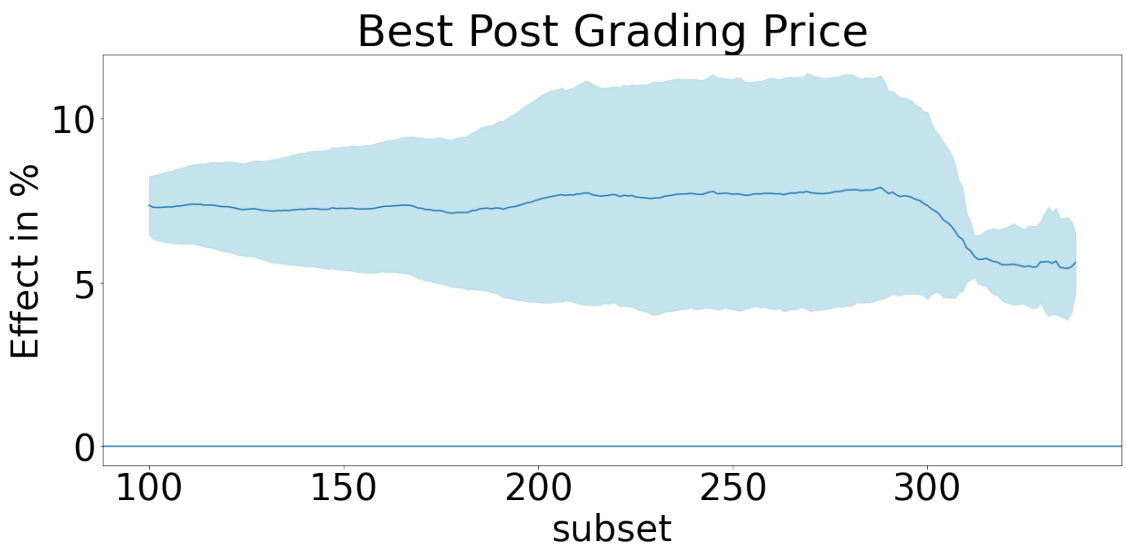
[163]:
```
plot_reg_rolling_perc_robust(regression_result_rolling_robust, 'Post Grading␣
 ↪Price')
plt.savefig(f"{fig_output_path}"␣
 ↪+"post_grading_price_rolling_regression_robust_perc.png")
```



[164]:
```
plot_reg_rolling_perc(regression_result_rolling, 'Best Post Grading Price␣
↪Viewed')
plt.savefig(f"{fig_output_path}"␣
↪+"best_post_grading_price_viewed_rolling_regression_perc.png")
```



[165]:
```
plot_reg_rolling_perc(regression_result_rolling, 'Best Post Grading Price')
plt.savefig(f"{fig_output_path}"␣
↪+"best_post_grading_price_rolling_regression_perc.png")
```



[166]:
```
users_receipt['post_grading_price_treatment_effect'] =␣
↪users_receipt['post_grading_price'] -␣
↪users_receipt['counterfactual_post_grading_price_mean']
```

[167]:
```
users_receipt['post_grading_price_treatment_effect_special_group_robust'] =␣
↪(users_receipt['post_grading_price'] -
                                                                    ␣
↪users_receipt['counterfactual_post_grading_price_special_group_robust']
                                                                    )
```

[168]:
```
users_receipt['best_price_viewed_treatment_effect'] =␣
↪users_receipt['best_price_viewed'] -␣
↪users_receipt['counterfactual_price_mean']
```

[169]:
```
#users_receipt['best_price_viewed_treatment_effect'].
↪fillna(users_receipt['price_treatment_effect'], inplace = True)
```

[170]:
```
users_receipt['best_price_treatment_effect'] = users_receipt['best_price'] -␣
↪users_receipt['counterfactual_price_mean']
```

[171]:
```
users_receipt['platform_age']=(users_receipt['date'] - users_receipt['date'].
↪min()).dt.days
```

[172]:
```
mask = users_receipt['post_grading_price_treatment_effect'].notna()

Y = users_receipt['post_grading_price_treatment_effect'][mask]
X = sm.add_constant(users_receipt['upload_order'][mask])
```

```
mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id'][mask]},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

[172]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                  OLS Regression Results
==============================================================================
==============
Dep. Variable:     post_grading_price_treatment_effect   R-squared:
0.021
Model:                                             OLS   Adj. R-squared:
0.020
Method:                                  Least Squares   F-statistic:
5.075
Date:                                 Sun, 29 Aug 2021   Prob (F-statistic):
0.0249
Time:                                         20:38:06   Log-Likelihood:
-26567.
No. Observations:                                5312   AIC:
5.314e+04
Df Residuals:                                    5310   BIC:
5.315e+04
Df Model:                                           1
Covariance Type:                              cluster
==============================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const            2.1474      2.197      0.977      0.328      -2.159       6.454
upload_order     0.0956      0.042      2.253      0.024       0.012       0.179
==============================================================================
Omnibus:                     2170.184   Durbin-Watson:                   1.633
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            53989.487
Skew:                           1.391   Prob(JB):                         0.00
Kurtosis:                      18.368   Cond. No.                         102.
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
```

```
"""
```

[173]:
```
mask = users_receipt['price_treatment_effect'].notna()

Y = users_receipt['price_treatment_effect'][mask]
X = sm.add_constant(users_receipt['upload_order'][mask])

mod = sm.OLS(Y,X.astype(float))

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id'][mask]},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

[173]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
==
Dep. Variable:     price_treatment_effect   R-squared:
0.005
Model:                                OLS   Adj. R-squared:
0.005
Method:                     Least Squares   F-statistic:
5.539
Date:                   Sun, 29 Aug 2021   Prob (F-statistic):
0.0191
Time:                           20:38:07   Log-Likelihood:
-28572.
No. Observations:                   5673   AIC:
5.715e+04
Df Residuals:                       5671   BIC:
5.716e+04
Df Model:                              1
Covariance Type:                 cluster
==============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const           2.7618      2.007      1.376      0.169      -1.172       6.696
upload_order    0.0499      0.021      2.353      0.019       0.008       0.091
==============================================================================
Omnibus:                     1711.604   Durbin-Watson:                   1.630
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           163429.996
Skew:                          -0.431   Prob(JB):                         0.00
Kurtosis:                      29.280   Cond. No.                         98.7
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

5.3 Rolling Regression New Buyer

[174]:
```
cumulative_new_buyer_fraction_final =␣
↪users_receipt[['upload_order','user_id','cumulative_new_buyer_fraction','special_price_memb
↪drop_duplicates(subset='upload_order',keep='last')
```

[175]:
```
pd.pivot_table(cumulative_new_buyer_fraction_final, index=␣
↪'special_price_member',
               values='cumulative_new_buyer_fraction',
               aggfunc=[np.mean,lambda x: stats.sem(x.dropna())])
```

[175]:
```
                                          mean  \
                      cumulative_new_buyer_fraction
special_price_member
False                                     0.690641
True                                      0.296481

                                          <lambda>
                      cumulative_new_buyer_fraction
special_price_member
False                                     0.020289
True                                      0.027197
```

[176]:
```
def get_rolling_new_buyer(data_used,max_upload_order, window_size):
    new_buyer_mean = []
    new_buyer_sem = []
    new_buyer_emp_5q = []
    new_buyer_emp_95q = []
    data_points = []
    users_subset = []
    subset = np.arange(window_size,max_upload_order)

    for i in subset:
        mask = ((data_used['upload_order']>i-window_size) &
                (data_used['upload_order']<=i) &
                (data_used['rolling_reg_subset'])
               )
```

```
        new_buyer_mean.append(data_used['cumulative_new_buyer_fraction'][mask].
↪mean())
        new_buyer_sem.append(stats.
↪sem(data_used['cumulative_new_buyer_fraction'][mask]))
        new_buyer_emp_5q.append(np.
↪quantile(data_used['cumulative_new_buyer_fraction'][mask],0.05))
        new_buyer_emp_95q.append(np.
↪quantile(data_used['cumulative_new_buyer_fraction'][mask],0.95))
        data_points.append(len(data_used['new_buyer'][mask]))
        users_subset.append(list(data_used['user_id'][mask].unique()))

    output = pd.DataFrame({'subset':subset,
                           'number_data_points':data_points,
                           'fraction_new_buyer':new_buyer_mean,
                           'fraction_new_buyer_sem':new_buyer_sem,
                           'fraction_new_buyer_5q':new_buyer_emp_5q,
                           'fraction_new_buyer_95q':new_buyer_emp_95q,
                           'users_subset':users_subset
                          })

    output['lower'] = output['fraction_new_buyer'] - 1.
↪96*output['fraction_new_buyer_sem']
    output['upper'] = output['fraction_new_buyer'] + 1.
↪96*output['fraction_new_buyer_sem']

    return output
```

[177]:
```
def get_rolling_new_buyer_all(data_used,max_upload_order,window_size):
    new_buyer_mean = []
    new_buyer_sem = []
    new_buyer_emp_5q = []
    new_buyer_emp_95q = []
    data_points = []
    users_subset = []
    subset = np.arange(window_size,max_upload_order)

    for i in subset:
        mask = ((data_used['upload_order']>i-window_size) &
                (data_used['upload_order']<=i) #&
                #(data_used['rolling_reg_subset'])
               )

        new_buyer_mean.append(data_used['cumulative_new_buyer_fraction'][mask].
↪mean())
        new_buyer_sem.append(stats.
↪sem(data_used['cumulative_new_buyer_fraction'][mask]))
```

```
        new_buyer_emp_5q.append(np.
↪quantile(data_used['cumulative_new_buyer_fraction'][mask],0.05))
        new_buyer_emp_95q.append(np.
↪quantile(data_used['cumulative_new_buyer_fraction'][mask],0.95))
        data_points.append(len(data_used['new_buyer'][mask]))
        users_subset.append(list(data_used['user_id'][mask].unique()))

    output = pd.DataFrame({'subset':subset,
                           'number_data_points':data_points,
                           'fraction_new_buyer':new_buyer_mean,
                           'fraction_new_buyer_sem':new_buyer_sem,
                           'fraction_new_buyer_5q':new_buyer_emp_5q,
                           'fraction_new_buyer_95q':new_buyer_emp_95q,
                           'users_subset':users_subset
                          })

    output['lower'] = output['fraction_new_buyer'] - 1.
↪96*output['fraction_new_buyer_sem']
    output['upper'] = output['fraction_new_buyer'] + 1.
↪96*output['fraction_new_buyer_sem']

    return output
```

[178]:
```
new_buyer_rolling = get_rolling_new_buyer(users_receipt,193,100)
```

[179]:
```
new_buyer_rolling_50 = get_rolling_new_buyer(users_receipt,193,50)
```

[180]:
```
new_buyer_rolling_75 = get_rolling_new_buyer(users_receipt,193,75)
```

[181]:
```
new_buyer_rolling_all = get_rolling_new_buyer_all(users_receipt,193,100)
```

5.3.1 Plotting new buyer rolling regression

[182]:
```
def plot_new_buyer(data, x, y):

    tmp_df = pd.pivot_table(data, index= x, values=y,
                            aggfunc=[np.nanmin,np.nanmean, np.nanmax,'count']
                           )
    tmp_df.reset_index(inplace=True)

    tmp_df.columns = ['index','min','mean','max','count']

    fig, ax = plt.subplots(figsize=(20,14))

    plt.rcParams.update({'font.size': 18})
```

```
    ax.plot(tmp_df['index'], tmp_df['mean'], color='black')

    #plt.fill_between(tmp_df['index'], tmp_df['max'],
    #                 tmp_df['min'],
    #                alpha=0.7, color='lightblue')

    scatter = ax.scatter(tmp_df['index'], tmp_df['mean'], color='black',
                         s = np.array(tmp_df['count'])
                         )

    #tmp_df.plot.scatter(['index'],['mean'],
    #                    s = tmp_df['count'],
    #                    color='black', ax = ax)

    # produce a legend with the unique colors from the scatter
    #legend1 = ax.legend(*scatter.legend_elements(),
    #                    loc="lower left", title="Classes")
    #ax.add_artist(legend1)

    # produce a legend with a cross section of sizes from the scatter
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
    legend2 = ax.legend(handles, labels, loc="lower right", title="Data Points")

    plt.xlabel('Upload Order')

    #plt.xlim(0,235)

    plt.ylim(0,1)

    plt.ylabel('Fraction New Buyer')
```

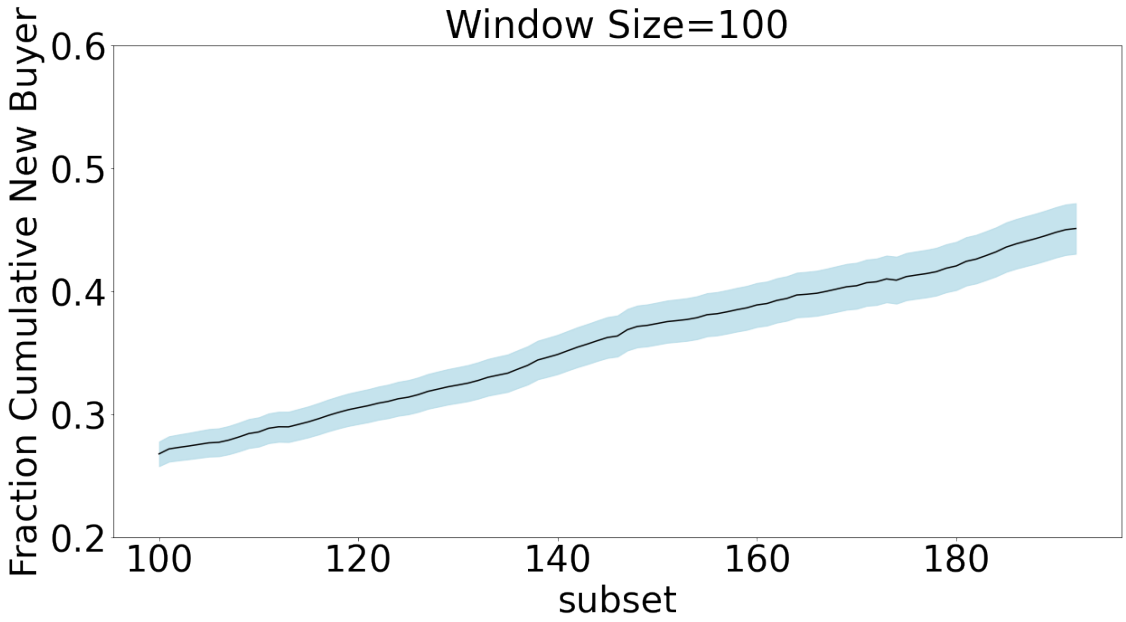
[183]:
```
def plot_new_buyer_rolling(data):

    tmp_df = data

    fig, ax = plt.subplots(figsize=(20,10))

    plt.rcParams.update({'font.size': 25})

    ax.plot(tmp_df['subset'], tmp_df['fraction_new_buyer'], color='black')

    scatter = ax.scatter(tmp_df['subset'], tmp_df['fraction_new_buyer'],␣
↪color='blue',
                         s = np.array(tmp_df['number_data_points']),alpha=0.2
                         )
```

```
    handles, labels = scatter.legend_elements(prop="sizes", alpha=0.
↪6,color='blue')
    legend2 = ax.legend(handles, labels, loc="lower right", title="Data Points")

    plt.xlabel('subset')

    #plt.xlim(0,235)

    plt.ylim(0,1)

    plt.ylabel('Fraction Cumulative New Buyer')
```

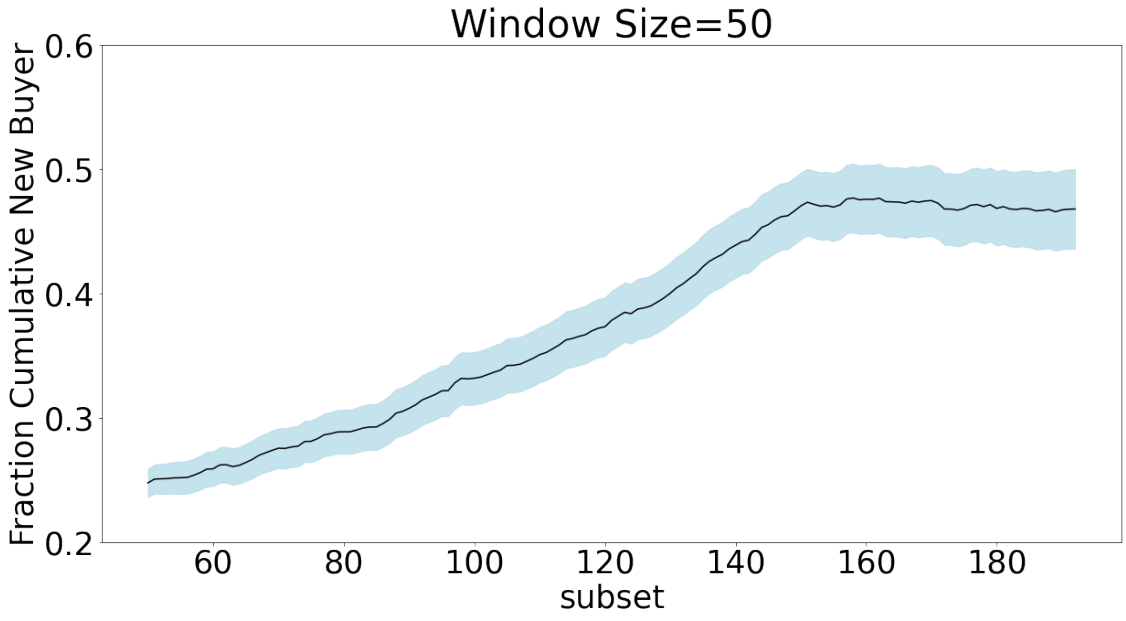
[184]:
```
def plot_new_buyer_rolling_ci(data,title):

    tmp_df = data

    fig, ax = plt.subplots(figsize=(20,10))

    plt.rcParams.update({'font.size': 35})

    ax.plot(tmp_df['subset'], tmp_df['fraction_new_buyer'], color='black')

    #scatter = ax.scatter(tmp_df['subset'], tmp_df['fraction_new_buyer'],␣
↪color='blue',
    #                     s = np.array(tmp_df['number_data_points']),alpha=0.2
    #                     )

    #plt.fill_between(tmp_df['subset'], tmp_df['fraction_new_buyer_95q'],
    #                 tmp_df['fraction_new_buyer_5q'],
    #                 alpha=0.2, color='red')

    plt.fill_between(tmp_df['subset'], tmp_df['upper'],
                     tmp_df['lower'],
                     alpha=0.7, color='lightblue')

    #handles, labels = scatter.legend_elements(prop="sizes", alpha=0.
↪6,color='blue')
    #legend2 = ax.legend(handles, labels, loc="lower right", title="Data␣
↪Points")

    plt.xlabel('subset')

    #plt.xlim(0,235)

    plt.ylim(0.2,0.6)
    plt.ylim(0.2,0.6)
```

```
    plt.ylabel('Fraction Cumulative New Buyer')

    plt.title(title)
```

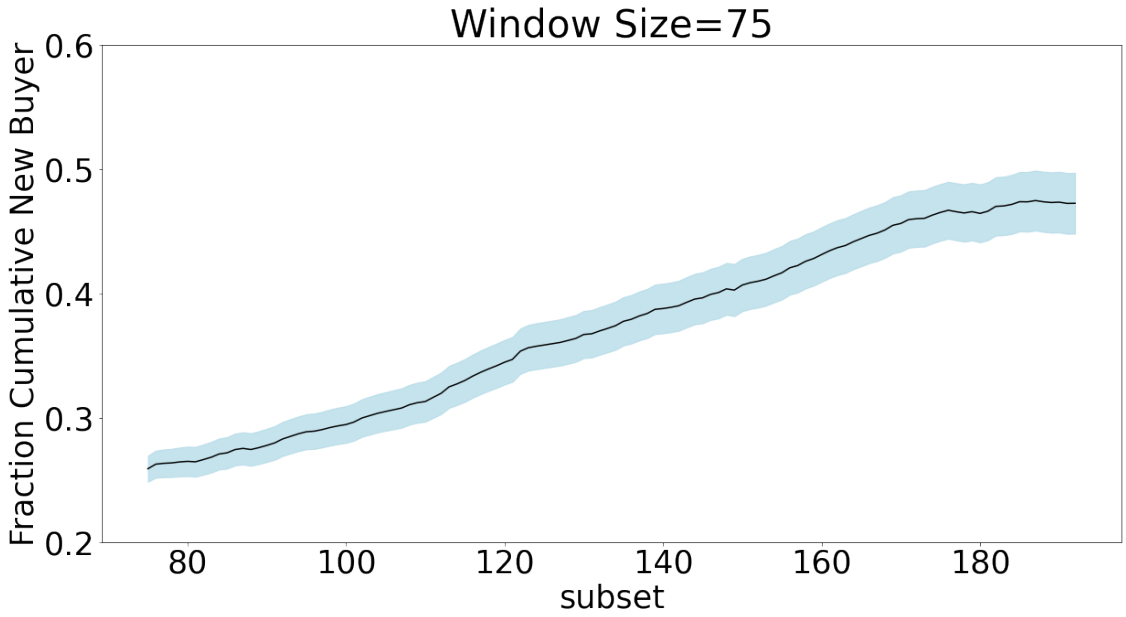
[185]:
```
plot_new_buyer_rolling_ci(new_buyer_rolling,"Window Size=100")
plt.savefig(f"{fig_output_path}" +"fraction_new_buyer_100_window_size.png")
```



[186]:
```
plot_new_buyer_rolling_ci(new_buyer_rolling_50,"Window Size=50")
plt.savefig(f"{fig_output_path}" +"fraction_new_buyer_50_window_size.png")
```



[187]:
```
plot_new_buyer_rolling_ci(new_buyer_rolling_75,"Window Size=75")
plt.savefig(f"{fig_output_path}" +"fraction_new_buyer_75_window_size.png")
```



5.4 Create Regression Table for HTE

5.4.1 Weekly Price CV

[188]:
```
counterfactual_price['isoweekday'] = pd.to_timedelta([row.isoweekday() for row␣
↪in counterfactual_price['date']], unit="d")

counterfactual_price['week_start'] = counterfactual_price['date']-␣
↪counterfactual_price['isoweekday']
```

[189]:
```
weekly_counterfactual_price = pd.pivot_table(counterfactual_price,
                                index = ['week_start','buyer','trans_type'],
                                values = 'price_counterfactual_table',
                                aggfunc=[np.nanmedian]
                               )

weekly_counterfactual_price.columns = ['price']

weekly_counterfactual_price.reset_index(inplace=True)

weekly_counterfactual_price
```

[189]:

	week_start	buyer	trans_type	price
0	2018-11-25	mill//6//13//nan	mill	975.0
1	2018-11-25	mill//6//29//nan	mill	975.0
2	2018-12-02	mill//1//18//nan	mill	902.5
3	2018-12-02	mill//1//21//nan	mill	900.0
4	2018-12-02	mill//1//24//nan	mill	902.5
…	…	…	…	…
5857	2021-02-07	mill//8//27//nan	mill	1905.0
5858	2021-02-07	mill//8//38//nan	mill	1905.0
5859	2021-02-07	mill//8//39//nan	mill	1905.0
5860	2021-02-07	mill//92//203//nan	mill	2020.0
5861	2021-02-07	mill//98//205//nan	mill	2070.0

[5862 rows x 4 columns]

[190]:
```
mask = weekly_counterfactual_price['trans_type']=='mill'

weekly_price_cv = pd.pivot_table(weekly_counterfactual_price[mask],
                                 index = ['week_start'],
                                 values = 'price',
                                 aggfunc=[np.nanmean,np.nanstd]
                                )

weekly_price_cv.columns = ['mean_price','std_price']

weekly_price_cv['coeff_var'] = weekly_price_cv['std_price']/
 ↪weekly_price_cv['mean_price']

weekly_price_cv.reset_index(inplace = True)

#weekly_price_cv['trans_type'].replace({'mill':'MILL','lr//':'RAMP'}, inplace =␣
 ↪True)

weekly_price_cv['week_start'] = pd.to_datetime(weekly_price_cv['week_start']).
 ↪dt.date

weekly_price_cv
```

[190]:

	week_start	mean_price	std_price	coeff_var
0	2018-11-25	975.000000	0.000000	0.000000
1	2018-12-02	980.138889	61.083791	0.062322
2	2018-12-09	1037.222222	75.463708	0.072756
3	2018-12-16	1100.972222	63.980344	0.058113
4	2018-12-23	1121.666667	56.827914	0.050664
..	…	…	…	…
111	2021-01-10	2002.875000	45.543548	0.022739
112	2021-01-17	1899.625000	44.363555	0.023354

```
113  2021-01-24  1897.312500  52.193512   0.027509
114  2021-01-31  1926.250000  54.449459   0.028267
115  2021-02-07  1933.092105  60.363298   0.031226

[116 rows x 4 columns]
```

[191]:
```
users_receipt['post_grading_price_treatment_effect'] =␣
↪users_receipt['post_grading_price'] -␣
↪users_receipt['counterfactual_post_grading_price_mean']
```

[192]:
```
village_nighttime_light_density =␣
↪users_receipt[['nighttime_light_density','kecamatan_desa']].
↪drop_duplicates().dropna()

village_nighttime_light_density
```

[192]:
```
      nighttime_light_density                 kecamatan_desa
0                    1.432564       BATANG GANSAL__BELIMBING
4                    0.837024         BATANG GANSAL__SIAMBUL
7                    2.778166      SEBERIDA__PANGKALAN KASAI
63                   1.200885              SEBERIDA__KELESA
72                   1.022429          BATANG GANSAL__RINGIN
…                       …                           …
7678                 1.283064              ENOK__PENGALIHAN
7693                 0.916706         KERITANG__TELUK KELASA
7717                 0.937358  RENGAT BARAT__SIALANG DUA DAHAN
7804                 0.771157         KELAYANG__TELUK SEJUAH
7858                 1.851617         KEMUNING__KEMUNING TUA

[130 rows x 2 columns]
```

[193]:
```
village_nighttime_light_density['nighttime_light_density_percentile']= (
    [stats.
↪percentileofscore(village_nighttime_light_density['nighttime_light_density'],row)
     for row in village_nighttime_light_density['nighttime_light_density']]
)
```

[194]:
```
village_nighttime_light_density['nighttime_light_density_top25'] =␣
↪1*(village_nighttime_light_density['nighttime_light_density_percentile']>75)
```

[195]:
```
village_nighttime_light_density['nighttime_light_density_top50'] =␣
↪1*(village_nighttime_light_density['nighttime_light_density_percentile']>50)
```

[196]:
```
village_nighttime_light_density['nighttime_light_density_bottom25'] =␣
↪1*(village_nighttime_light_density['nighttime_light_density_percentile']<25)
```

[197]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(village_nighttime_light_density.
 ↪drop(columns = ['nighttime_light_density']),
                                     on = 'kecamatan_desa', how ='left'
                                    )

print(users_receipt.shape)
```

```
(7933, 290)
(7933, 294)
```

[198]:
```
users_receipt['price_treatment_effect'] = users_receipt['price_imputed_final']␣
 ↪- users_receipt['counterfactual_price_mean']
```

[199]:
```
users_receipt['price_treatment_effect'].describe()
```

[199]:
```
count    5673.000000
mean        5.211565
std        37.348673
min      -543.750000
25%         0.000000
50%         0.000000
75%         0.000000
max       265.000000
Name: price_treatment_effect, dtype: float64
```

[200]:
```
kecamatan_dummies = pd.get_dummies(users['kecamatan'],prefix='kd')
```

[201]:
```
village_dummies = pd.get_dummies(users['kecamatan_desa'],prefix='vd')
```

[202]:
```
user_dummies = pd.get_dummies(users['user_id'],prefix='ud')
```

[203]:
```
print(users.shape)

users = users.merge(kecamatan_dummies, left_index=True, right_index= True,␣
 ↪how='left')

print(users.shape)
```

```
(2060, 51)
(2060, 81)
```

[204]:
```
print(users.shape)

users = users.merge(village_dummies, left_index=True, right_index= True,␣
 ↪how='left')
```

```
print(users.shape)
```

```
(2060, 81)
(2060, 290)
```

[205]:
```
print(users.shape)

users = users.merge(user_dummies, left_index=True, right_index= True,␣
↪how='left')

print(users.shape)
```

```
(2060, 290)
(2060, 2350)
```

[206]:
```
dummy_village = [col for col in users.columns if col.startswith('vd_')]
```

[207]:
```
dummy_kecamatan = [col for col in users.columns if col.startswith('kd_')]

cols = ['user_id'] + dummy_kecamatan

print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols], on ='user_id', how='left')

print(users_receipt.shape)
```

```
(7933, 294)
(7933, 324)
```

[208]:
```
dummy_village = [col for col in users.columns if col.startswith('vd_')]

cols = ['user_id'] + dummy_village

print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols], on ='user_id', how='left')

print(users_receipt.shape)
```

```
(7933, 324)
(7933, 533)
```

[209]:
```
dummy_user = [col for col in users.columns if col.startswith('ud_')]

cols = ['user_id'] + dummy_user
```

```
print(users_receipt.shape)

users_receipt = users_receipt.merge(users[cols], on ='user_id', how='left')

print(users_receipt.shape)
```

```
(7933, 533)
(7933, 2593)
```

[210]:
```
cols = [col for col in users_receipt if col.startswith('dummy_special_group')]

users_receipt[cols] = users_receipt[cols].fillna(0)
```

[211]:
```
users_receipt['binary_role'].value_counts()
```

[211]:
```
non_farmer    6477
farmer        1456
Name: binary_role, dtype: int64
```

[212]:
```
users_receipt['is_non_farmer'] = (users_receipt['binary_role']=='non_farmer')*1
```

[213]:
```
users_receipt['is_non_farmer'].value_counts()
```

[213]:
```
1    6477
0    1456
Name: is_non_farmer, dtype: int64
```

[214]:
```
(users_receipt['binary_role'] + users_receipt['special_price_member'].
↪astype(str)).unique()
```

[214]:
```
array(['non_farmerFalse', 'non_farmerTrue', 'farmerFalse', 'farmerTrue'],
      dtype=object)
```

[215]:
```
users_receipt['user_type'] = users_receipt['binary_role'] +␣
↪users_receipt['special_price_member'].astype(str)
```

[216]:
```
users_receipt['user_type'].unique()
```

[216]:
```
array(['non_farmerFalse', 'non_farmerTrue', 'farmerFalse', 'farmerTrue'],
      dtype=object)
```

[217]:
```
users_receipt['user_type'].replace({'farmerFalse':'farmer_wo_special_group',
                                    'farmerTrue':'farmer_with_special_group',
                                    'non_farmerFalse':
↪'non_farmer_wo_special_group',
                                    'non_farmerTrue':
↪'non_farmer_with_special_group',
```

```
                                    }, inplace = True)
```

[218]:
```
users_receipt['user_type'].value_counts()
```

[218]:
```
non_farmer_wo_special_group      4395
non_farmer_with_special_group    2082
farmer_wo_special_group          1034
farmer_with_special_group         422
Name: user_type, dtype: int64
```

[219]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(pd.
 ↪get_dummies(users_receipt['user_type'],prefix='ut'),
                                    left_index=True, right_index=True, how =␣
 ↪'left'
                                    )

print(users_receipt.shape)
```

```
(7933, 2595)
(7933, 2599)
```

[220]:
```
users_receipt['role'].value_counts()
```

[220]:
```
Middle Man       6075
Amprah Farmer    1456
Ramp Manager      402
Name: role, dtype: int64
```

[221]:
```
users_receipt['is_farmer'] = (users_receipt['role']=='Amprah Farmer')*1
```

[222]:
```
users_receipt['is_middleman'] = (users_receipt['role']=='Middle Man')*1
```

[223]:
```
users_receipt['is_rampmanager'] = (users_receipt['role']=='Ramp Manager')*1
```

[224]:
```
users_receipt['special_price_member_dummy'] =␣
 ↪users_receipt['special_price_member']*1
```

[225]:
```
users_receipt['month_year'] = pd.to_datetime(users_receipt['date']).dt.
 ↪strftime("%Y-%m")
```

[226]:
```
month_dummies = pd.get_dummies(users_receipt['month_year'],prefix='md')

month_dummies
```

[226]:

	md_2018-12	md_2019-01	md_2019-02	md_2019-03	md_2019-04	md_2019-05	\
0	1	0	0	0	0	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	1	0	0	0	0	0	
4	1	0	0	0	0	0	
…	…	…	…	…	…	…	
7928	0	0	0	0	0	0	
7929	0	0	0	0	0	0	
7930	0	0	0	0	0	0	
7931	0	0	0	0	0	0	
7932	0	0	0	0	0	0	

	md_2019-06	md_2019-07	md_2019-08	md_2019-09	…	md_2020-05	\
0	0	0	0	0	…	0	
1	0	0	0	0	…	0	
2	0	0	0	0	…	0	
3	0	0	0	0	…	0	
4	0	0	0	0	…	0	
…	…	…	…	…	…	…	
7928	0	0	0	0	…	0	
7929	0	0	0	0	…	0	
7930	0	0	0	0	…	0	
7931	0	0	0	0	…	0	
7932	0	0	0	0	…	0	

	md_2020-06	md_2020-07	md_2020-08	md_2020-09	md_2020-10	md_2020-11	\
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	0	
…	…	…	…	…	…	…	
7928	0	0	0	0	0	0	
7929	0	0	0	0	0	0	
7930	0	0	0	0	0	0	
7931	0	0	0	0	0	0	
7932	0	0	0	0	0	0	

	md_2020-12	md_2021-01	md_2021-02
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
…	…	…	…
7928	1	0	0

```
7929          1          0          0
7930          0          1          0
7931          0          1          0
7932          0          1          0

[7933 rows x 27 columns]
```

[227]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(month_dummies,left_index=True ,
                                    right_index=True, how='left')

print(users_receipt.shape)
```

```
(7933, 2604)
(7933, 2631)
```

[228]:
```
mask = counterfactual_price['price_cleaned'].notna()

daily_unique_price_shared = pd.pivot_table(counterfactual_price[mask],␣
↪index='date', values='buyer',
                                           aggfunc=lambda x: len(x.unique()))

daily_unique_price_shared.columns = ['number_unique_price_shared_daily']

daily_unique_price_shared.reset_index(inplace=True)
```

[229]:
```
print(users_receipt.shape)

users_receipt = users_receipt.merge(daily_unique_price_shared,on ='date' ,␣
↪how='left')

print(users_receipt.shape)
```

```
(7933, 2631)
(7933, 2632)
```

[230]:
```
daily_price_cv = pd.pivot_table(counterfactual_price,
                                index = ['date','trans_type'],
                                values = 'price_cleaned',
                                aggfunc=[np.nanmean,np.nanstd]
                               )

daily_price_cv.columns = ['mean_price','std_price']

daily_price_cv['coeff_var'] = daily_price_cv['std_price']/
↪daily_price_cv['mean_price']
```

```
daily_price_cv.reset_index(inplace = True)

daily_price_cv['trans_type'].replace({'mill':'MILL','lr//':'RAMP'}, inplace =␣
↪True)

daily_price_cv['date'] = pd.to_datetime(daily_price_cv['date']).dt.date
```

[231]:
```
daily_price_cv.head()
```

[231]:

	date	trans_type	mean_price	std_price	coeff_var
0	2018-12-02	MILL	975.00	NaN	NaN
1	2018-12-03	MILL	983.75	61.288253	0.062301
2	2018-12-04	MILL	992.50	102.530483	0.103305
3	2018-12-05	MILL	997.50	52.360927	0.052492
4	2018-12-06	MILL	997.50	52.360927	0.052492

[232]:
```
users_receipt['isoweekday'] = pd.to_timedelta([row.isoweekday() for row in␣
↪users_receipt['date']], unit="d")

users_receipt['week_start'] = users_receipt['date']- users_receipt['isoweekday']
```

[233]:
```
print(users_receipt.shape)

cols= ['week_start','coeff_var']

#mask = weekly_price_cv['trans_type']=='MILL'

users_receipt = users_receipt.merge(weekly_price_cv[cols],on ='week_start' ,␣
↪how='left')

print(users_receipt.shape)
```

```
(7933, 2634)
(7933, 2635)
```

[234]:
```
daily_price_cv_all = pd.pivot_table(counterfactual_price,
                                    index = ['date'],
                                    values = 'price_cleaned',
                                    aggfunc=[np.nanmean,np.nanstd]
                                   )

daily_price_cv_all.columns = ['mean_price','std_price']

daily_price_cv_all['coeff_var_all'] = daily_price_cv_all['std_price']/
↪daily_price_cv_all['mean_price']
```

```
daily_price_cv_all.reset_index(inplace = True)

#daily_price_cv_all['trans_type'].replace({'mill':'MILL','lr//':'RAMP'},␣
↪inplace = True)

daily_price_cv_all['date'] = pd.to_datetime(daily_price_cv_all['date']).dt.date
```

[235]:
```
print(users_receipt.shape)

cols= ['date','coeff_var_all']

users_receipt = users_receipt.merge(daily_price_cv_all[cols],on ='date' ,␣
↪how='left')

print(users_receipt.shape)
```

```
(7933, 2635)
(7933, 2636)
```

[236]:
```
#mask = old_buyers_df['user_id'].isin(list(regression_table_hte_pgp['user_id'].
↪unique()))

old_buyers_breakdown = pd.pivot_table(old_buyers_df, index = 'user_id',␣
↪columns='trans_type', values = 'buyer',
                                      aggfunc=lambda x:len(x.unique())
                                     ).reset_index().fillna(0)

old_buyers_breakdown['mill_ratio_old_buyers'] = old_buyers_breakdown['mill']/
↪(old_buyers_breakdown['mill']+ old_buyers_breakdown['lr'])

old_buyers_breakdown.head()
```

[236]:

trans_type	user_id	lr	mill	mill_ratio_old_buyers
0	100	1.0	2.0	0.666667
1	1000	0.0	1.0	1.000000
2	1001	0.0	1.0	1.000000
3	1002	1.0	1.0	0.500000
4	1003	0.0	1.0	1.000000

[237]:
```
print(users_receipt.shape)

cols= ['user_id','mill_ratio_old_buyers']

users_receipt = users_receipt.merge(old_buyers_breakdown[cols],on ='user_id' ,␣
↪how='left')

print(users_receipt.shape)
```

```
(7933, 2636)
(7933, 2637)
```

[238]:
```python
def create_reg_table_post_grading_price_hte():

    tmp_df_1 = create_regression_table(users_receipt, 'post_grading_price',
                                       'counterfactual_post_grading_price_mean')

    dummy_kecamatan = [col for col in users_receipt.columns if col.
↪startswith('kd_')]

    dummy_village = [col for col in users_receipt.columns if col.
↪startswith('vd_')]

    dummy_month = [col for col in users_receipt.columns if col.
↪startswith('md_')]

    #dummy_user_type = [col for col in users_receipt.columns if col.
↪startswith('ut_')]

    #dummy_user = [col for col in users.columns if col.startswith('ud_')]


    cols = ['receipt_id','user_id',
            ␣
↪'kecamatan_desa',"special_price_member","post_grading_price_treatment_effect",
             'upload_order','mean_daily_mill_price','unique_daily_active_user',
             #'user_density_village','buyer_density_village',
             'platform_age','new_buyer',
             'is_farmer', 'is_middleman', 'is_rampmanager','is_non_farmer',
          ␣
↪'special_price_member_dummy','coeff_var',#'coeff_var_all',#'mill_ratio_old_buyers',
             #'nighttime_light_density_top50',
            ␣
↪'number_unique_price_shared_daily','num_local_price_available_5village',
            'num_local_price_available_8village'
           #,'nighttime_light_density_bottom25'
            ] + dummy_village +dummy_month + dummy_kecamatan

    tmp_df = tmp_df_1.merge(users_receipt[cols],
                            on =['receipt_id','user_id'],
                            how='left'
                           )

    tmp_df = tmp_df.dropna()

    print(tmp_df.shape)
```

```
   covariates =␣
↪['upload_order','mean_daily_mill_price','unique_daily_active_user',
                 #'user_density_village','buyer_density_village',
                 #'platform_age',
                 'is_farmer', 'is_middleman', 'is_rampmanager','is_non_farmer',
                ␣
↪'special_price_member_dummy','coeff_var',#'coeff_var_all',#'mill_ratio_old_buyers',
                 #'nighttime_light_density_top50',
                ␣
↪'number_unique_price_shared_daily','num_local_price_available_5village',
            'num_local_price_available_8village'
                ␣
↪#'nighttime_light_density_top25','nighttime_light_density_bottom25'
                 #,'nighttime_light','nighttime_light_density'
                ]

   interact_vars = covariates + dummy_village + dummy_month + dummy_kecamatan

   tmp_df_interact = tmp_df[interact_vars]

   #print(tmp_df_interact['is_farmer'])

   #print(tmp_df_interact['special_price_member_dummy'])

   for col in tmp_df_interact.columns:
       #print(col)
       tmp_df_interact[col] = tmp_df['treatment_dummy']*tmp_df_interact[col]

   tmp_df_interact.columns = ['tic_' + col if col in covariates
                              #else 'tiud_' + col if col in dummy_user
                              else 'tiv_' + col if col in dummy_village
                              else 'tim_' + col if col in dummy_month
                              else 'tik_' + col if col in dummy_kecamatan
                              #else 'tisp_' + col if col in dummy_special_group
                              else col
                              for col in tmp_df_interact.columns
                             ]

   tmp_df.columns = ['c_' + col if col in covariates
                     #else 'ud_' + col if col in dummy_user
                     else 'v_' + col if col in dummy_village
                     else 'm_' + col if col in dummy_month
                     else 'k_' + col if col in dummy_kecamatan
                     #else 'sp_' + col if col in dummy_special_group
                     else col
                     for col in tmp_df.columns
```

```
                    ]

    tmp_df = tmp_df.merge(tmp_df_interact, left_index= True, right_index=True,␣
↪how ='left')

    tmp_df['new_buyer_rate'] = tmp_df['new_buyer']*tmp_df['treatment_dummy']

    #tmp_df.loc[:, (tmp_df != tmp_df.iloc[0]).any()]

    for col in tmp_df.columns:
        #print(col)
        if len(tmp_df[col].unique())==1:
            tmp_df.drop(columns=[col], inplace=True)

    return tmp_df
```

[239]:
```
regression_table_hte_pgp= create_reg_table_post_grading_price_hte()
regression_table_hte_pgp.head()
```

```
(9534, 288)
```

[239]:

	receipt_id	user_id	variable	post_grading_price	\
190	1757	31	post_grading_price	1195.740	
191	1762	77	post_grading_price	1189.160	
192	1765	77	post_grading_price	1160.392	
193	1769	31	post_grading_price	1196.390	
194	1780	89	post_grading_price	1174.650	

	treatment_dummy	kecamatan_desa	special_price_member	\
190	1	SEBERIDA__PANGKALAN KASAI	False	
191	1	BATANG GANSAL__BELIMBING	False	
192	1	BATANG GANSAL__BELIMBING	False	
193	1	SEBERIDA__PANGKALAN KASAI	False	
194	1	BATANG GANSAL__BELIMBING	False	

	post_grading_price_treatment_effect	c_upload_order	\
190	0.000000	27	
191	1.577500	106	
192	-28.740188	107	
193	0.000000	28	
194	0.000000	14	

	c_mean_daily_mill_price	…	tik_kd_PERANAP	tik_kd_RAKIT KULIM	\
190	-243.136201	…	0	0	
191	-243.136201	…	0	0	
192	-241.878509	…	0	0	
193	-241.878509	…	0	0	

```
194              -246.136201  …               0                    0

     tik_kd_RENGAT  tik_kd_RENGAT BARAT  tik_kd_RETEH  tik_kd_SEBERIDA  \
190              0                    0             0                1
191              0                    0             0                0
192              0                    0             0                0
193              0                    0             0                1
194              0                    0             0                0

     tik_kd_SUNGAI LALA  tik_kd_TEMPULING  tik_kd_UKUI  new_buyer_rate
190                   0                 0            0               0
191                   0                 0            0               1
192                   0                 0            0               1
193                   0                 0            0               0
194                   0                 0            0               0

[5 rows x 357 columns]
```

[240]:
```
regression_table_hte_pgp.shape
```

[240]:
```
(9534, 357)
```

[241]:
```
regression_table_hte_pgp.to_excel(f"{output_path}" + "regression_table_hte_pgp.
↪xlsx")
```

6 Subhypothesis Analysis

6.1 Seller's Learning

[242]:
```
users_receipt['upload_order_high'] = (users_receipt['upload_order'] >= ␣
↪users_receipt['upload_order'].median())*1
```

[243]:
```
mask = users_receipt['post_grading_price_treatment_effect'].notna()

Y = users_receipt['post_grading_price_treatment_effect'][mask]
X = sm.add_constant(users_receipt['upload_order_high'][mask])

mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id'][mask]},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')
```

```
results.summary()
```

[243]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                  OLS Regression Results
==============================================================================
===============
Dep. Variable:     post_grading_price_treatment_effect   R-squared:
0.006
Model:                                             OLS   Adj. R-squared:
0.006
Method:                                  Least Squares   F-statistic:
2.769
Date:                                 Sun, 29 Aug 2021   Prob (F-statistic):
0.0970
Time:                                         20:39:06   Log-Likelihood:
-26605.
No. Observations:                                5312   AIC:
5.321e+04
Df Residuals:                                    5310   BIC:
5.323e+04
Df Model:                                           1
Covariance Type:                              cluster
==============================================================================
=====
                         coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                  4.1530      2.389      1.739      0.082      -0.529
8.834
upload_order_high      5.8027      3.487      1.664      0.096      -1.032
12.638
==============================================================================
Omnibus:                     2164.232   Durbin-Watson:                   1.639
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            51910.300
Skew:                           1.396   Prob(JB):                         0.00
Kurtosis:                      18.058   Cond. No.                         2.60
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

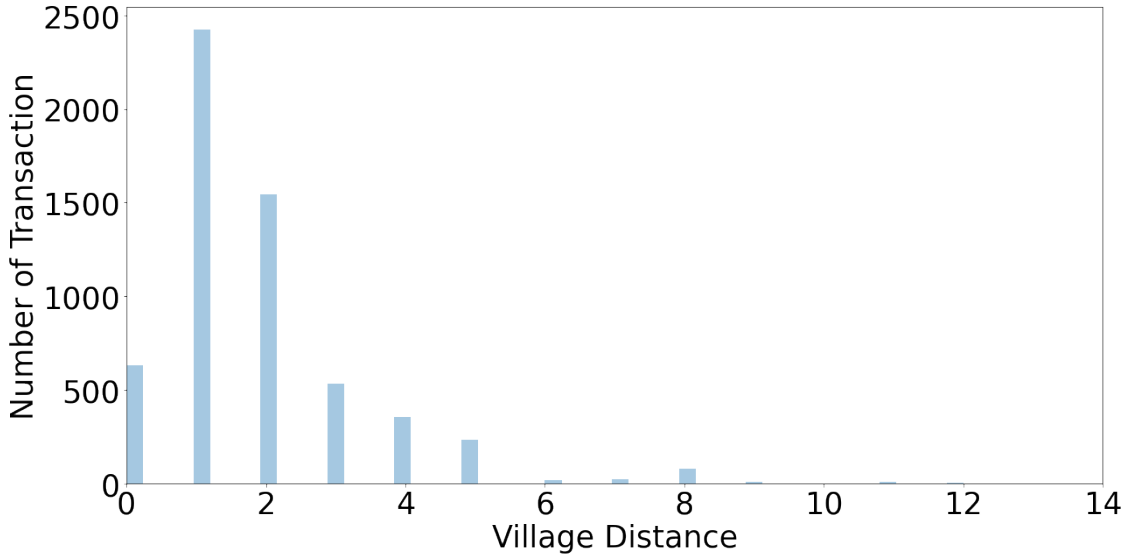
[244]:
```
Y = users_receipt['new_buyer']
X = sm.add_constant(users_receipt['upload_order_high'])
```

```
mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id']},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

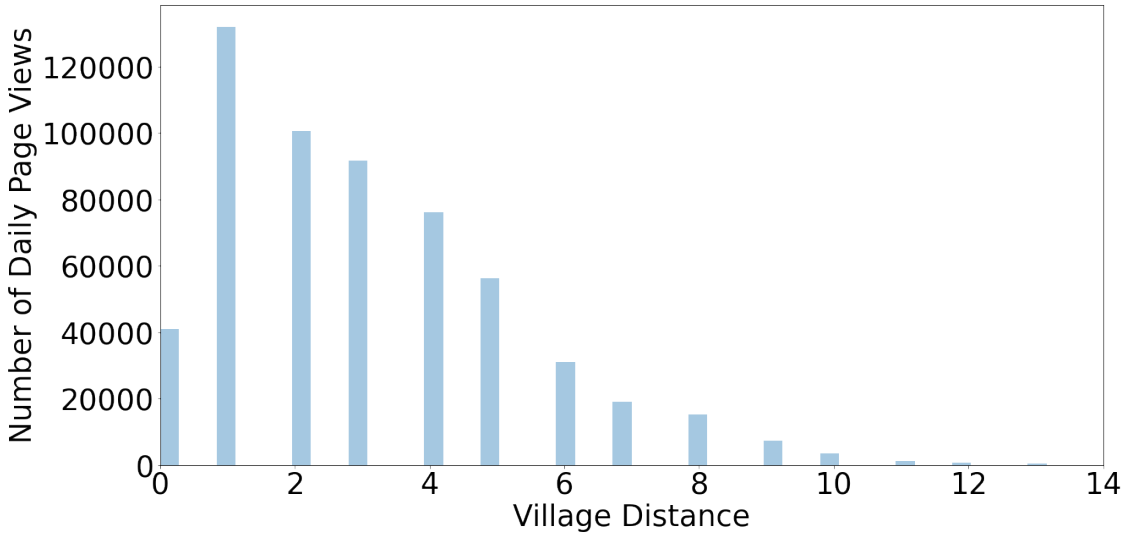
[244]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              new_buyer   R-squared:                       0.020
Model:                            OLS   Adj. R-squared:                  0.020
Method:                 Least Squares   F-statistic:                     4.012
Date:                Sun, 29 Aug 2021   Prob (F-statistic):             0.0457
Time:                        20:39:06   Log-Likelihood:                -5675.4
No. Observations:                7933   AIC:                         1.135e+04
Df Residuals:                    7931   BIC:                         1.137e+04
Df Model:                           1
Covariance Type:              cluster
==============================================================================
=====
                         coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                  0.4396      0.037     11.827      0.000       0.367
0.512
upload_order_high      0.1416      0.071      2.003      0.045       0.003
0.280
==============================================================================
Omnibus:                    28790.727   Durbin-Watson:                   1.685
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1216.119
Skew:                          -0.043   Prob(JB):                    8.38e-265
Kurtosis:                       1.084   Cond. No.                         2.63
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

6.2 Local Price Information

[245]:
```
fig, ax = plt.subplots(figsize=(20, 10))
sns.distplot(users_receipt['shortest_path_length'].dropna(), kde=False, ax = ax)

plt.xlim(0,14)
plt.xlabel('Village Distance')
plt.ylabel('Number of Transaction')
plt.savefig(f"{fig_output_path}" +"transaction_distance.png")
```



[246]:
```
fig, ax = plt.subplots(figsize=(20, 10))
sns.distplot(do_ramp_visit['shortest_path_length'].dropna(), kde=False, ax = ax)

plt.xlim(0,14)
plt.xlabel('Village Distance')
plt.ylabel('Number of Daily Page Views')
plt.savefig(f"{fig_output_path}" +"browsing_distance.png")
```



[247]:
```
users_receipt['local_price_available_high'] =␣
↪(users_receipt['num_local_price_available_5village'] >= ␣
↪users_receipt['num_local_price_available_5village'].median())*1
```

[248]:
```
mask = users_receipt['post_grading_price_treatment_effect'].notna()

Y = users_receipt['post_grading_price_treatment_effect'][mask]
X = sm.add_constant(users_receipt['local_price_available_high'][mask])

mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id'][mask]},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

[248]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                OLS Regression Results
==============================================================================
==============
Dep. Variable:     post_grading_price_treatment_effect   R-squared:
0.000
Model:                                             OLS   Adj. R-squared:
```

```
-0.000
Method:                                 Least Squares   F-statistic:
0.009226
Date:                                Sun, 29 Aug 2021   Prob (F-statistic):
0.924
Time:                                        20:39:07   Log-Likelihood:
-26622.
No. Observations:                                5312   AIC:
5.325e+04
Df Residuals:                                    5310   BIC:
5.326e+04
Df Model:                                           1
Covariance Type:                              cluster
==============================================================================
==============
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
--------------
const                          6.8565      3.081      2.225      0.026
0.817      12.896
local_price_available_high     0.3479      3.622      0.096      0.923
-6.752       7.448
==============================================================================
Omnibus:                     2121.643   Durbin-Watson:                   1.638
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            51750.772
Skew:                           1.352   Prob(JB):                         0.00
Kurtosis:                      18.050   Cond. No.                         2.50
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

[249]:
```
mask_bottom = ((users_receipt['local_price_available_high']==0) &
               (users_receipt['post_grading_price_treatment_effect'] >-300)
              )

mask_top = ((users_receipt['local_price_available_high']==1) &
            (users_receipt['post_grading_price_treatment_effect'] >-300)
              )

print(np.mean(users_receipt['post_grading_price_treatment_effect'][mask_bottom].
 ↪dropna())-
      np.mean(users_receipt['post_grading_price_treatment_effect'][mask_top].
 ↪dropna())
                     )
```

```
scipy.stats.
↪ttest_ind(users_receipt['post_grading_price_treatment_effect'][mask_bottom].
↪dropna(),
                    ␣
↪users_receipt['post_grading_price_treatment_effect'][mask_top].dropna()
                    )
```

```
-0.48542812995111984
```

[249]:
```
Ttest_indResult(statistic=-0.4862787203966994, pvalue=0.6267896357644729)
```

[250]:
```
#mask = users_receipt['post_grading_price_treatment_effect'] >-300

Y = users_receipt['new_buyer']
X = sm.add_constant(users_receipt['local_price_available_high'])

mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id']},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

[250]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              new_buyer   R-squared:                       0.025
Model:                            OLS   Adj. R-squared:                  0.025
Method:                 Least Squares   F-statistic:                     4.264
Date:                Sun, 29 Aug 2021   Prob (F-statistic):             0.0394
Time:                        20:39:07   Log-Likelihood:                -5653.5
No. Observations:                7933   AIC:                         1.131e+04
Df Residuals:                    7931   BIC:                         1.132e+04
Df Model:                           1
Covariance Type:              cluster
==============================================================================
==============
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
```

```
--------------
const                           0.4346      0.057      7.681      0.000
0.324       0.545
local_price_available_high      0.1597      0.077      2.065      0.039
0.008       0.311
==============================================================================
Omnibus:                    29227.284   Durbin-Watson:                   1.665
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1187.718
Skew:                          -0.042   Prob(JB):                    1.23e-258
Kurtosis:                       1.106   Cond. No.                         2.57
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

[251]:
```
mask_bottom = ((users_receipt['local_price_available_high']==0) #&
               #(users_receipt['post_grading_price_treatment_effect'] >-300)
              )

mask_top = ((users_receipt['local_price_available_high']==1) #&
            #(users_receipt['post_grading_price_treatment_effect'] >-300)
              )

print(np.mean(users_receipt['new_buyer'][mask_bottom].dropna())-
      np.mean(users_receipt['new_buyer'][mask_top].dropna())
                    )

scipy.stats.ttest_ind(users_receipt['new_buyer'][mask_bottom].dropna(),
                      users_receipt['new_buyer'][mask_top].dropna()
                     )
```

```
-0.1596773074099332
```

[251]:
```
Ttest_indResult(statistic=-14.395326975059147, pvalue=2.1214409540686373e-46)
```

[252]:
```
mask_bottom = ((users_receipt['local_price_available_high']==0) #&
               #(users_receipt['post_grading_price_treatment_effect'] >-300)
              )

mask_top = ((users_receipt['local_price_available_high']==1) #&
            #(users_receipt['post_grading_price_treatment_effect'] >-300)
              )

print(np.mean(users_receipt['shortest_path_length'][mask_bottom].dropna())-
      np.mean(users_receipt['shortest_path_length'][mask_top].dropna())
                    )
```

```
scipy.stats.ttest_ind(users_receipt['shortest_path_length'][mask_bottom].
↪dropna(),
                      users_receipt['shortest_path_length'][mask_top].dropna()
                     )
```

```
0.5981282138585509
```

[252]:
```
Ttest_indResult(statistic=15.172158665558968, pvalue=4.9900429069349923e-51)
```

[253]:
```
users_receipt.groupby(['role'])['upload_order'].max()
```

[253]:
```
role
Amprah Farmer    130
Middle Man       372
Ramp Manager     101
Name: upload_order, dtype: int64
```

6.3 Special Price Membership

[254]:
```
mask = users_receipt['post_grading_price_treatment_effect'].notna()

Y = users_receipt['post_grading_price_treatment_effect'][mask]
X = sm.add_constant(users_receipt['special_price_member_dummy'][mask])

mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id'][mask]},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')

results.summary()
```

[254]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 OLS Regression Results
==============================================================================
===============
Dep. Variable:     post_grading_price_treatment_effect   R-squared:
0.003
Model:                                             OLS   Adj. R-squared:
0.003
Method:                                  Least Squares   F-statistic:
```

```
0.9344
Date:                                Sun, 29 Aug 2021   Prob (F-statistic):
0.334
Time:                                        20:39:08   Log-Likelihood:
-26614.
No. Observations:                            5312   AIC:
5.323e+04
Df Residuals:                                5310   BIC:
5.324e+04
Df Model:                                       1
Covariance Type:                          cluster
==============================================================================
==============
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
--------------
const                           8.6628      3.880      2.233      0.026
1.058      16.268
special_price_member_dummy     -4.1729      4.317     -0.967      0.334
-12.634       4.288
==============================================================================
Omnibus:                     2084.157   Durbin-Watson:                   1.643
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            50448.911
Skew:                           1.320   Prob(JB):                         0.00
Kurtosis:                      17.865   Cond. No.                         2.45
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

[255]:
```
#mask = users_receipt['post_grading_price_treatment_effect'] >-300

Y = users_receipt['new_buyer']
X = sm.add_constant(users_receipt['special_price_member_dummy'])

mod = sm.OLS(Y,X)

results = mod.fit(cov_type='cluster',
              cov_kwds={'groups': users_receipt['user_id']},
              use_correction = True,
                  missing='drop'
             )

#print('Best Post Grading Price Viewed Rolling Regression (50-150)')
```

```
results.summary()
```

[255]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              new_buyer   R-squared:                       0.057
Model:                            OLS   Adj. R-squared:                  0.057
Method:                 Least Squares   F-statistic:                     9.084
Date:                Sun, 29 Aug 2021   Prob (F-statistic):            0.00271
Time:                        20:39:08   Log-Likelihood:                -5522.3
No. Observations:                7933   AIC:                         1.105e+04
Df Residuals:                    7931   BIC:                         1.106e+04
Df Model:                           1
Covariance Type:              cluster
==============================================================================
==============
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
--------------
const                          0.5922      0.062      9.612      0.000
0.471       0.713
special_price_member_dummy    -0.2571      0.085     -3.014      0.003
-0.424      -0.090
==============================================================================
Omnibus:                    32545.290   Durbin-Watson:                   1.691
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1028.710
Skew:                          -0.064   Prob(JB):                    4.15e-224
Kurtosis:                       1.240   Cond. No.                         2.42
==============================================================================

Notes:
[1] Standard Errors are robust to cluster correlation (cluster)
"""
```

7 Users Level Analysis

[256]:
```
users_receipt['post_grading_weight_final'].describe()
```

[256]:
```
count     7316.000000
mean      4335.919696
std       2735.775458
min         20.000000
25%       1919.255500
50%       3970.210000
```

```
75%       6301.500000
max      19400.000000
Name: post_grading_weight_final, dtype: float64
```

[257]:
```
mask = users_receipt['post_grading_weight_final'] > 20000

users_receipt[['weight_source','post_grading_weight_final']][mask]
```

[257]:
```
Empty DataFrame
Columns: [weight_source, post_grading_weight_final]
Index: []
```

[258]:
```
mask = users_receipt['post_grading_weight_final'] <0

users_receipt[['weight_source','post_grading_weight_final']][mask]
```

[258]:
```
Empty DataFrame
Columns: [weight_source, post_grading_weight_final]
Index: []
```

[259]:
```
users_stats = pd.pivot_table(users_receipt,
               index = ['user_id','role','binary_role'
                       ],
               values = 'post_grading_price_difference_%',
               aggfunc=['count',np.nanmean,np.nanmin,np.nanmedian,np.nanmax,np.
 ↪nanstd]
              )

users_stats.columns = [ i[0] + ' ' + i[1]for i in users_stats.columns
                      ]
```

[260]:
```
users_stats.reset_index(inplace=True)

users_stats.shape
```

[260]:
```
(513, 9)
```

[261]:
```
users_receipt['revenue'].describe()
```

[261]:
```
count    7.216000e+03
mean     7.097566e+06
std      4.680388e+06
min      2.960000e+04
25%      3.299580e+06
50%      6.325148e+06
75%      9.861902e+06
max      3.910990e+07
```

```
Name: revenue, dtype: float64
```

[262]:
```
users_receipt['counterfactual_revenue_mean'].describe()
```

[262]:
```
count    5.285000e+03
mean     7.215246e+06
std      4.960855e+06
min      2.960000e+04
25%      3.138300e+06
50%      6.341242e+06
75%      1.024760e+07
max      3.910990e+07
Name: counterfactual_revenue_mean, dtype: float64
```

[263]:
```
mask = (~(users_receipt['revenue'].isna())&
        ~(users_receipt['counterfactual_revenue_mean'].isna())
       )

users_revenue = pd.pivot_table(users_receipt[mask],
                               index = 'user_id',
                               values =␣
 ↪['revenue','counterfactual_revenue_mean'],
                               aggfunc=[np.sum,'count']
                              )

#users_revenue = users_revenue.replace({0:np.nan}).dropna()

#users_revenue['delta_revenue'] = (users_revenue['revenue'] -␣
 ↪users_revenue['counterfactual_revenue'])

users_revenue.shape
```

[263]:
```
(320, 4)
```

[264]:
```
users_revenue.columns =␣
 ↪['counterfactual_revenue_mean','revenue','transaction_count','transaction_count_2']

users_revenue.head()
```

[264]:

user_id	counterfactual_revenue_mean	revenue	transaction_count	\
100	1.313902e+09	1.311524e+09	165	
1000	1.903841e+09	1.915213e+09	130	
1004	3.573820e+07	3.636220e+07	3	
1011	1.537615e+07	1.537615e+07	3	
1015	9.068798e+06	9.068798e+06	1	

```
         transaction_count_2
user_id
100                      165
1000                     130
1004                       3
1011                       3
1015                       1
```

[265]:
```
users_revenue.drop(columns=['transaction_count_2'], inplace = True)
```

[266]:
```
users_revenue.head()
```

[266]:
```
         counterfactual_revenue_mean       revenue  transaction_count
user_id
100                     1.313902e+09  1.311524e+09                165
1000                    1.903841e+09  1.915213e+09                130
1004                    3.573820e+07  3.636220e+07                  3
1011                    1.537615e+07  1.537615e+07                  3
1015                    9.068798e+06  9.068798e+06                  1
```

[267]:
```
users_revenue['delta_revenue'] = (users_revenue['revenue'] -␣
↪users_revenue['counterfactual_revenue_mean'])

users_revenue['delta_revenue'].describe()
```

[267]:
```
count    3.200000e+02
mean     5.429895e+05
std      4.193052e+06
min     -7.691911e+06
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      5.271621e+07
Name: delta_revenue, dtype: float64
```

[268]:
```
users_revenue['delta_revenue'].mean()
```

[268]:
```
542989.4803415118
```

[269]:
```
stats.sem(users_revenue['delta_revenue'].dropna())
```

[269]:
```
234398.74204042132
```

[270]:
```
print(users_stats.shape)

users_stats = users_stats.merge(users_revenue,
                                on='user_id',
```

```
                                    how='outer'
                                   )

print(users_stats.shape)
```

```
(513, 9)
(513, 13)
```

[271]:
```
users_stats['delta_revenue'].describe()
```

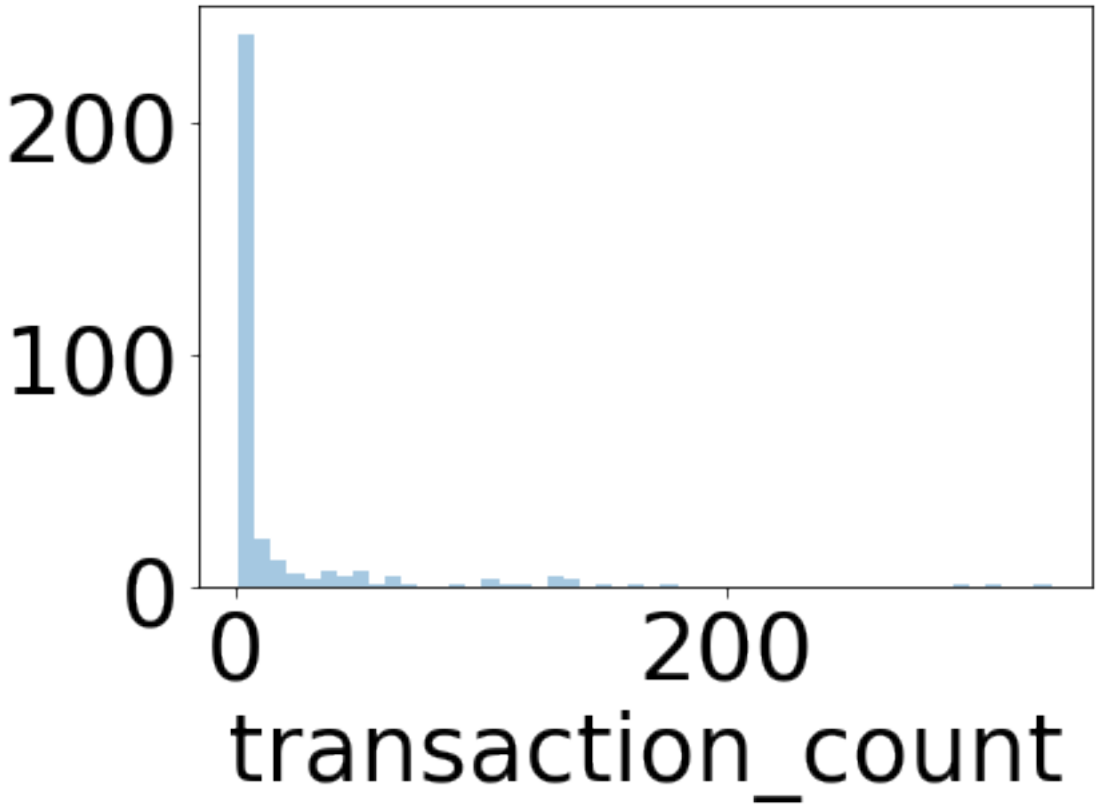
[271]:
```
count    3.200000e+02
mean     5.429895e+05
std      4.193052e+06
min     -7.691911e+06
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      5.271621e+07
Name: delta_revenue, dtype: float64
```

[272]:
```
users_stats['transaction_count'].describe()
```

[272]:
```
count    320.000000
mean      16.431250
std       41.763721
min        1.000000
25%        1.000000
50%        2.000000
75%        8.000000
max      333.000000
Name: transaction_count, dtype: float64
```

[273]:
```
sns.distplot(users_stats['transaction_count'].dropna(), kde=False)
```

[273]:
```
<AxesSubplot:xlabel='transaction_count'>
```



[274]:
```
users_stats['delta_revenue'].describe()
```

[274]:
```
count    3.200000e+02
mean     5.429895e+05
std      4.193052e+06
min     -7.691911e+06
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      5.271621e+07
Name: delta_revenue, dtype: float64
```

[275]:
```
users_stats['delta_revenue'].mean()
```

[275]:
```
542989.4803415118
```

[276]:
```
stats.sem(users_stats['delta_revenue'].dropna())
```

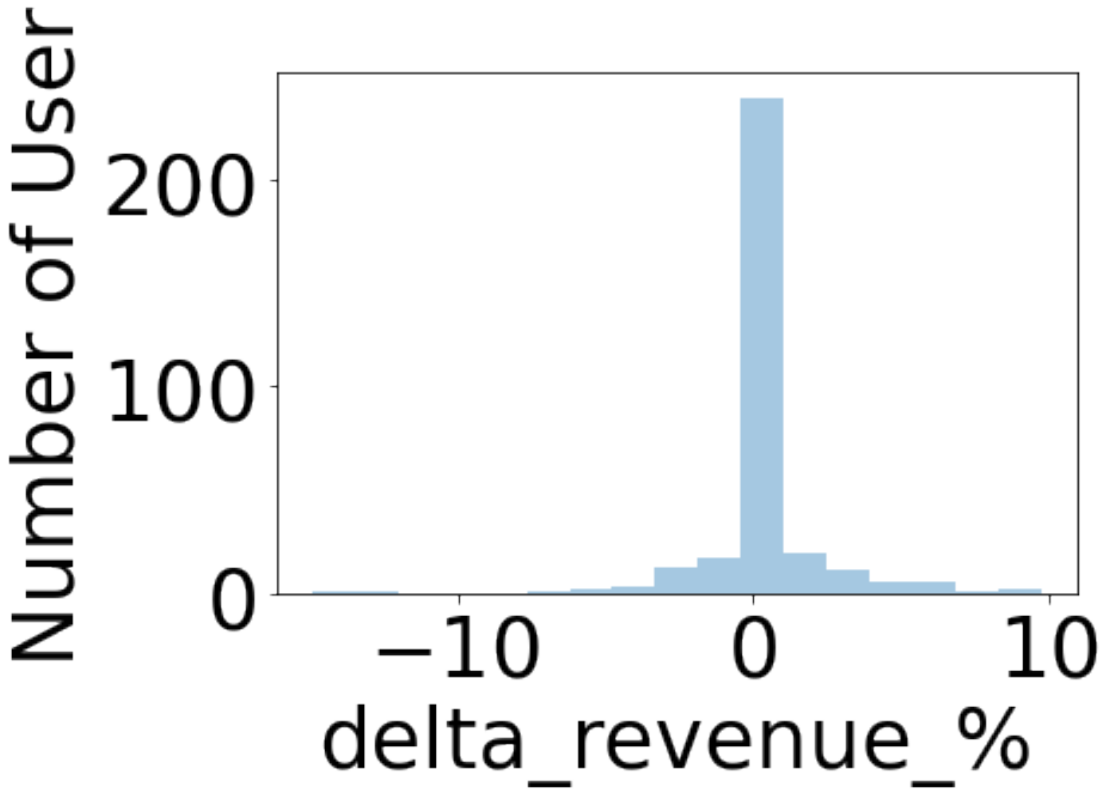
[276]:
```
234398.74204042132
```

[277]:
```
users_stats['delta_revenue_%'] = (users_stats['delta_revenue']/
 ↪users_stats['counterfactual_revenue_mean'])*100

users_stats['delta_revenue_%'].describe()
```

[277]:
```
count    320.000000
mean       0.158027
std        2.030335
min      -14.893617
25%        0.000000
50%        0.000000
75%        0.000000
max        9.714286
Name: delta_revenue_%, dtype: float64
```

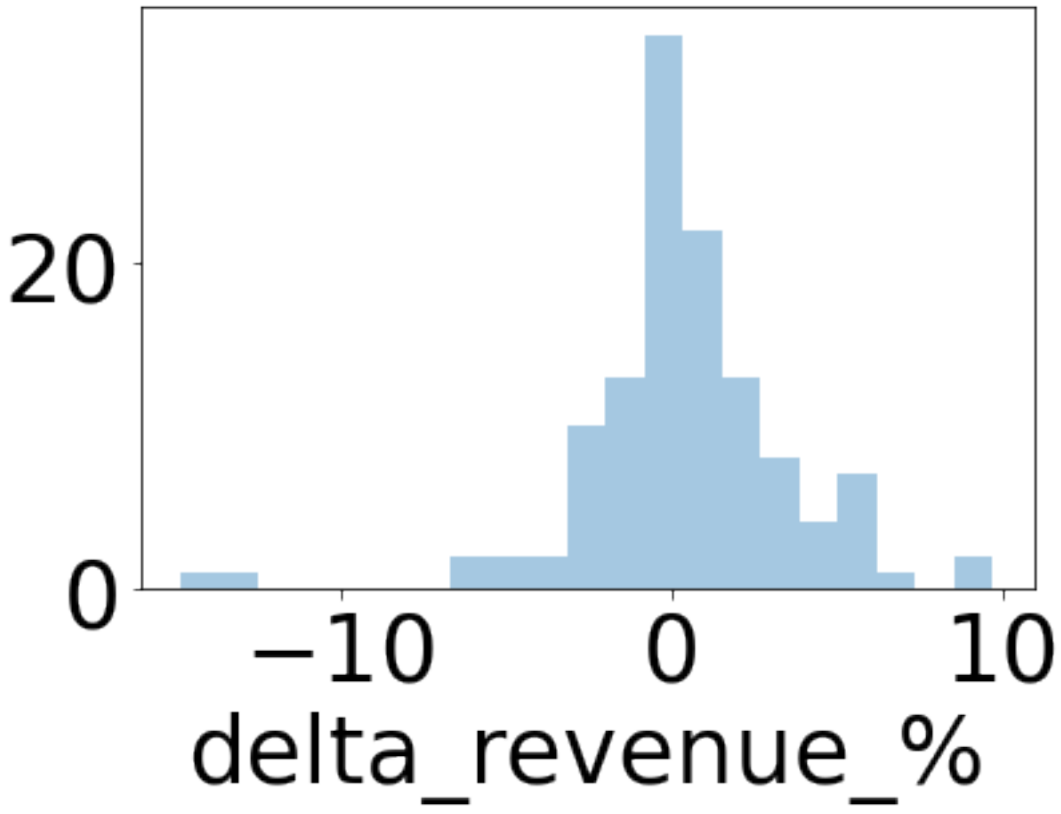
[278]:
```
mask = users_stats['delta_revenue_%'] <100

sns.distplot(users_stats['delta_revenue_%'].dropna(), kde= False)

plt.ylabel('Number of User')
```

[278]:
```
Text(0, 0.5, 'Number of User')
```



[279]:
```
mask = users_stats['delta_revenue_%'] !=0

sns.distplot(users_stats['delta_revenue_%'][mask].dropna(), kde= False)
```

[279]:
```
<AxesSubplot:xlabel='delta_revenue_%'>
```



[280]:
```
mask = (~(users_receipt['revenue'].isna())&
        ~(users_receipt['counterfactual_revenue_mean'].isna()) &
        (users_receipt['new_buyer'] ==True)
        #(users_receipt['revenue'] !=users_receipt['counterfactual_revenue'])
       )

users_revenue_new_buyer = pd.pivot_table(users_receipt[mask],
                                index = ['user_id','role'],
                                values =␣
 ↪['revenue','counterfactual_revenue_mean'],
                                aggfunc=np.sum
                               )

#users_revenue = users_revenue.replace({0:np.nan}).dropna()
```

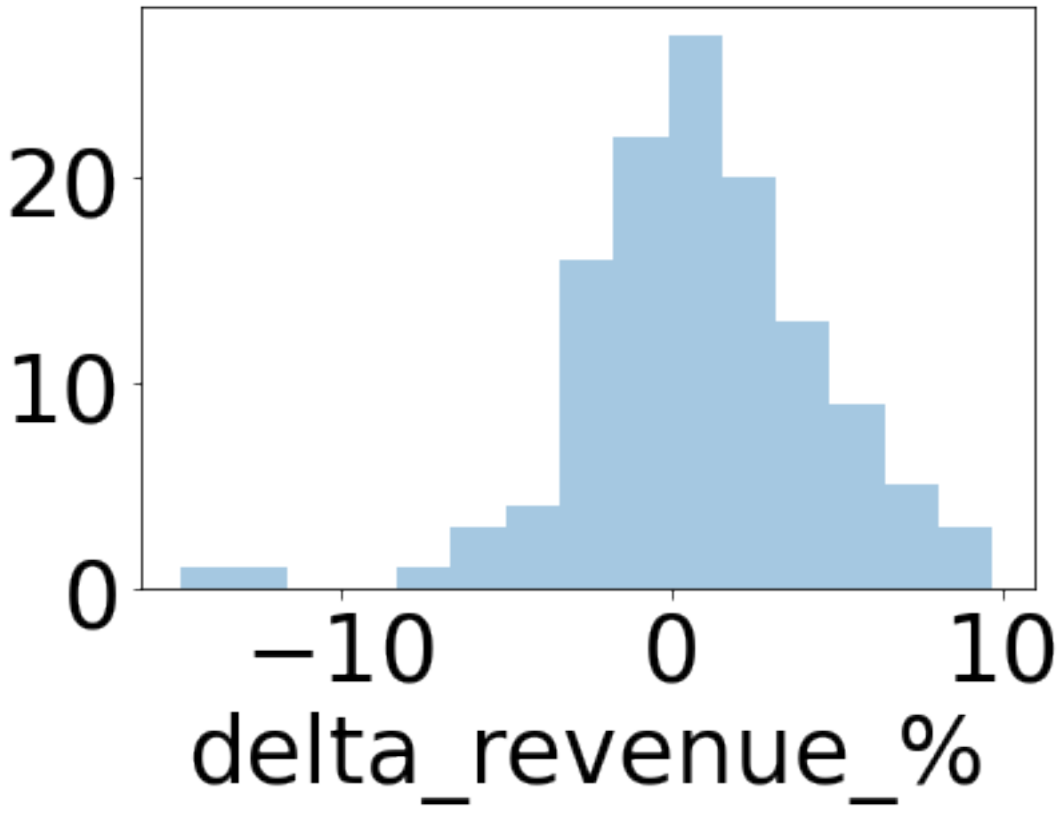
```
#users_revenue['delta_revenue'] = (users_revenue['revenue'] -␣
↪users_revenue['counterfactual_revenue'])

users_revenue_new_buyer.shape
```

[280]:
```
(125, 2)
```

[281]:
```
users_revenue_new_buyer['delta_revenue'] = (users_revenue_new_buyer['revenue'] -
                                            ␣
↪users_revenue_new_buyer['counterfactual_revenue_mean'])

users_revenue_new_buyer['delta_revenue_%'] =␣
↪users_revenue_new_buyer['delta_revenue']/
↪users_revenue_new_buyer['counterfactual_revenue_mean']*100
```

[282]:
```
sns.distplot(users_revenue_new_buyer['delta_revenue_%'].dropna(), kde= False)
```

[282]:
```
<AxesSubplot:xlabel='delta_revenue_%'>
```



[283]:
```
users_revenue_new_buyer.groupby('role')['delta_revenue_%'].mean()
```

[283]:
```
role
Amprah Farmer   -0.107502
Middle Man       1.280985
Ramp Manager     4.896960
Name: delta_revenue_%, dtype: float64
```

[284]:
```
users_revenue_new_buyer.groupby('role')['delta_revenue_%'].std()
```

[284]:
```
role
Amprah Farmer    3.578592
Middle Man       3.876912
Ramp Manager          NaN
Name: delta_revenue_%, dtype: float64
```

[285]:
```
print('Delta Revenue New Buyer Only')

pd.pivot_table(users_revenue_new_buyer,
               index = 'role',
               values = 'delta_revenue_%',
               aggfunc=['count',np.nanmean,lambda x: stats.sem(x.dropna()),
                     np.nanmedian,np.nanmin, np.nanmax,np.nanstd],
               margins = True
              )
```

```
Delta Revenue New Buyer Only
```

[285]:

	count	nanmean	<lambda>	nanmedian	nanmin	nanmax	nanstd
	delta_revenue_%	delta_revenue_%	delta_revenue_%	delta_revenue_%	delta_revenue_%	delta_revenue_%	delta_revenue_%
role							
Amprah Farmer	46	-0.107502	0.527635	-0.067476	-12.820513	7.356948	3.578592
Middle Man	78	1.280985	0.438974	0.879059	-14.893617	9.714286	3.876912
Ramp Manager	1	4.896960	NaN	4.896960	4.896960	4.896960	NaN
All	125	0.798950	0.341377	0.555556	-14.893617	9.714286	3.801413

[286]:
```
mask = users_stats['delta_revenue_%'] <100

users_stats['delta_revenue_%'][mask].describe()
```

[286]:
```
count    320.000000
mean       0.158027
std        2.030335
min      -14.893617
25%        0.000000
50%        0.000000
75%        0.000000
max        9.714286
Name: delta_revenue_%, dtype: float64
```

[287]:
```
mask = users_receipt['user_id']=='1814'

cols = ['revenue','counterfactual_revenue_mean','weight_source',
        'date','post_grading_weight_final','price_imputed_final',
        'counterfactual_price_mean'
       ]

users_receipt[cols][mask]
```

[287]:
```
        revenue  counterfactual_revenue_mean weight_source        date  \
6639  3700000.0                          NaN    fruit_sale  2021-01-14

      post_grading_weight_final  price_imputed_final  \
6639                     1850.0               2000.0

      counterfactual_price_mean
6639                        NaN
```

[288]:
```
mask = users_receipt['user_id']=='1456'

cols = ['revenue','counterfactual_revenue_mean','weight_source',
        'date','post_grading_weight_final','price_imputed_final',
        'counterfactual_price_mean'
       ]

users_receipt[cols][mask]
```

[288]:
```
        revenue  counterfactual_revenue_mean weight_source        date  \
4357        NaN                          NaN           NaN  2020-10-07
5648  2300800.0                    2703440.0    fruit_sale  2020-12-05
7861        NaN                          NaN           NaN  2020-10-08
7911        NaN                          NaN           NaN  2020-11-05

      post_grading_weight_final  price_imputed_final  \
4357                        NaN               1200.0
5648                     1438.0               1600.0
7861                        NaN                  NaN
```

```
7911                        NaN                   NaN

      counterfactual_price_mean
4357                     1200.0
5648                     1880.0
7861                        NaN
7911                     1800.0
```

[289]:
```
mask = users_receipt['user_id']=='1809'

cols = ['revenue','counterfactual_revenue_mean','weight_source',
        'date','post_grading_weight_final','price_imputed_final',
        'counterfactual_price_mean'
       ]

users_receipt[cols][mask]
```

[289]:
```
        revenue  counterfactual_revenue_mean weight_source        date  \
6525  3800000.0                          NaN    fruit_sale  2021-01-11
6559  1700000.0                    1950000.0    fruit_sale  2021-01-12

      post_grading_weight_final  price_imputed_final  \
6525                     1900.0               2000.0
6559                     1000.0               1700.0

      counterfactual_price_mean
6525                        NaN
6559                     1950.0
```

[290]:
```
mask = users_stats['delta_revenue_%'] >100

users_stats[mask]
```

[290]:
```
Empty DataFrame
Columns: [user_id, role, binary_role, count post_grading_price_difference_%,
nanmean post_grading_price_difference_%, nanmin post_grading_price_difference_%,
nanmedian post_grading_price_difference_%, nanmax
post_grading_price_difference_%, nanstd post_grading_price_difference_%,
counterfactual_revenue_mean, revenue, transaction_count, delta_revenue,
delta_revenue_%]
Index: []
```

[291]:
```
print('Delta Revenue % per User')

mask = ~users_stats['user_id'].isin(['1814']) ##user_id '1814' is an outlier␣
 ↪behaviours
```

```
delta_revenue_per_user = pd.pivot_table(users_stats[mask],
                                        index= 'binary_role',
                                        margins=True,
                                        values='delta_revenue_%',
                                        aggfunc=['count',np.nanmean,lambda x:␣
 ↪stats.sem(x.dropna()),
                                                 np.nanmedian,np.nanmin, np.
 ↪nanmax,np.nanstd]
                                        )

delta_revenue_per_user.columns =␣
 ↪['Count','Mean','SEM','Median','Min','Max','Std Dev']

#delta_revenue_per_user['null_hypothesis'] = 0

delta_revenue_per_user.reset_index(inplace = True)

delta_revenue_per_user
```

Delta Revenue % per User

[291]:

	binary_role	Count	Mean	SEM	Median	Min	Max	Std Dev
0	farmer	151	-0.016679	0.137056	0.0	-12.820513	7.356948	1.684167
1	non_farmer	169	0.314125	0.176134	0.0	-14.893617	9.714286	2.289743
2	All	320	0.158027	0.113499	0.0	-14.893617	9.714286	2.027160

[292]:
```
def get_pvalue_tstats(data_1):
    output = scipy.stats.ttest_1samp(data_1.dropna(),0)
    return output[1]
```

[293]:
```
get_pvalue_tstats(users_stats['delta_revenue_%'].dropna())
```

[293]: 0.1647956314805968

[294]:
```
delta_revenue_per_user_pvalue_list = []

for role in delta_revenue_per_user['binary_role']:
    print(role)
    if role =='All':
        p_value = get_pvalue_tstats(users_stats['delta_revenue_%'])
    else:
        mask = users_stats['binary_role']==role
```

```
        p_value = get_pvalue_tstats(users_stats['delta_revenue_%'][mask])
    delta_revenue_per_user_pvalue_list.append(p_value)

delta_revenue_per_user['p_value'] = delta_revenue_per_user_pvalue_list
```

```
farmer
non_farmer
All
```

[295]:
```
delta_revenue_per_user
```

[295]:

	binary_role	Count	Mean	SEM	Median	Min	Max	Std Dev	p_value
0	farmer	151	-0.016679	0.137056	0.0	-12.820513	7.356948	1.684167	0.903306
1	non_farmer	169	0.314125	0.176134	0.0	-14.893617	9.714286	2.289743	0.076320
2	All	320	0.158027	0.113499	0.0	-14.893617	9.714286	2.027160	0.164796

[296]:
```
delta_revenue_per_user['binary_role'].replace({'All':'All Role',
                                               'farmer':'Farmer',
                                               'non_farmer':'Middle Man & Ramp␣
↪Manager',
                                              }, inplace=True)
```

[297]:
```
delta_revenue_per_user['latex'] = ("\\" + "\\" +
                                   delta_revenue_per_user['binary_role'] + " &␣
↪" +
                                   delta_revenue_per_user['Count'].map(lambda x:
↪ '{0:.0f}'.format(x)).astype(str) + " & " +
                                   delta_revenue_per_user['Mean'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) + " & " +
                                   delta_revenue_per_user['SEM'].map(lambda x:␣
↪'{0:.3f}'.format(x)).astype(str) + " & " +
                                   delta_revenue_per_user['p_value'].map(lambda␣
↪x: '{0:.3f}'.format(x)).astype(str) +
                                   "\\" + "\\"
                                  )
```

[298]:
```
last_upload_by_user = users_receipt[['user_id','upload_order','date']].
↪groupby(['user_id'])[['upload_order','date']].max()
```

[299]:
```
last_upload_by_user.columns = ['last_upload_order','last_upload_date']
```

[300]:
```
first_upload_by_user = users_receipt[['user_id','upload_order','date']].
↪groupby(['user_id'])[['upload_order','date']].min()
```

[301]:
```
first_upload_by_user.columns = ['first_upload_order','first_upload_date']
```

[302]:
```
print(users_stats.shape)

users_stats = users_stats.merge(last_upload_by_user, on ='user_id',how ='left')

print(users_stats.shape)
```

```
(513, 14)
(513, 16)
```

[303]:
```
print(users_stats.shape)

users_stats = users_stats.merge(first_upload_by_user, on ='user_id',how ='left')

print(users_stats.shape)
```

```
(513, 16)
(513, 18)
```

[304]:
```
users_stats['first_last_upload_date_difference'] =␣
↪users_stats['last_upload_date'] - users_stats['first_upload_date']
```

[305]:
```
users_stats['average_upload_period'] =␣
↪(users_stats['first_last_upload_date_difference'].dt.days/
                                                  ␣
↪(users_stats['last_upload_order']-users_stats['first_upload_order'])
                                                  )
```

[306]:
```
users_stats.groupby(['binary_role'])['average_upload_period'].mean()
```

[306]:
```
binary_role
farmer        26.901603
non_farmer    22.119622
Name: average_upload_period, dtype: float64
```

[307]:
```
users_stats.groupby(['role'])['average_upload_period'].mean()
```

[307]:
```
role
Amprah Farmer    26.901603
Middle Man       21.359664
Ramp Manager     33.603431
Name: average_upload_period, dtype: float64
```

[308]:
```
print('Average Upload Period (in Days)')

mask = ~users_stats['user_id'].isin(['1814']) ##user_id '1814' is an outlier␣
↪behaviours

average_upload_period_per_user = pd.pivot_table(users_stats[mask],
                                        index= 'binary_role',
                                        margins=True,
                                        values='average_upload_period',
                                        aggfunc=['count',np.nanmean,lambda x:␣
↪stats.sem(x.dropna()),
                                                 np.nanmedian,np.nanmin, np.
↪nanmax,np.nanstd]
                                        )

average_upload_period_per_user.columns =␣
↪['Count','Mean','SEM','Median','Min','Max','Std Dev']

#delta_revenue_per_user['null_hypothesis'] = 0

average_upload_period_per_user.reset_index(inplace = True)

average_upload_period_per_user
```

Average Upload Period (in Days)

[308]:

	binary_role	Count	Mean	SEM	Median	Min	Max	Std Dev
0	farmer	94	26.901603	4.074585	11.321429	1.0	226.0	39.504563
1	non_farmer	145	22.119622	2.600510	7.181818	1.0	156.5	31.314282
2	All	239	24.000401	2.248749	8.916667	1.0	226.0	34.692011

[309]:
```
average_upload_period_per_user['binary_role'].replace({'All':'All Role',
                                                'farmer':'Farmer',
                                                'non_farmer':'Middle Man & Ramp␣
↪Manager',
                                               }, inplace=True)
```

[310]:
```
average_upload_period_per_user['latex'] = ("\\" + "\\" +
                                    ␣
↪average_upload_period_per_user['binary_role'] + " & " +
                                     average_upload_period_per_user['Count'].
↪map(lambda x: '{0:.0f}'.format(x)).astype(str) + " & " +
                                     average_upload_period_per_user['Mean'].
↪map(lambda x: '{0:.3f}'.format(x)).astype(str) + " & " +
                                     average_upload_period_per_user['SEM'].
↪map(lambda x: '{0:.3f}'.format(x)).astype(str) + #" & " +
```

```
                                    #delta_revenue_per_user['p_value'].
↪map(lambda x: '{0:.3f}'.format(x)).astype(str) +
                                    "\\" + "\\"
                                   )
```

[311]:
```
average_upload_period_per_user['latex'].loc[1]
```

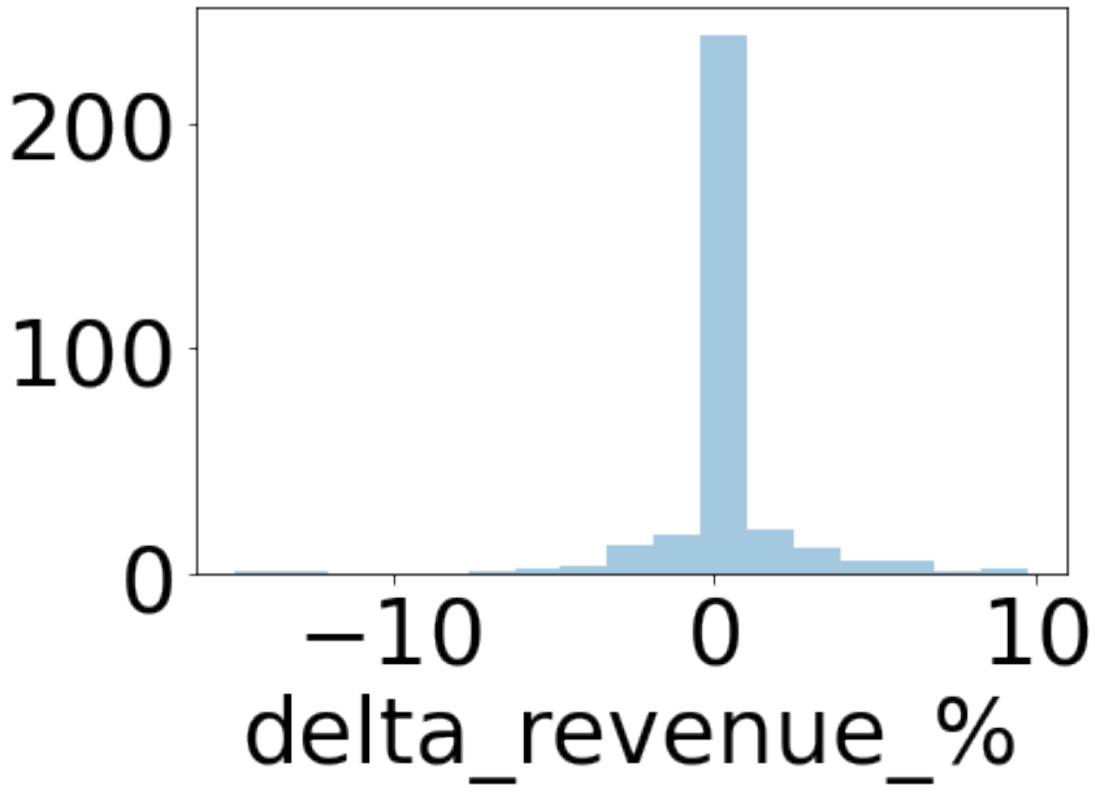
[311]:
```
'\\\\Middle Man & Ramp Manager & 145 & 22.120 & 2.601\\\\'
```

[312]:
```
users_stats['delta_revenue_%'][mask].describe()
```

[312]:
```
count    320.000000
mean       0.158027
std        2.030335
min      -14.893617
25%        0.000000
50%        0.000000
75%        0.000000
max        9.714286
Name: delta_revenue_%, dtype: float64
```

[313]:
```
mask = ~users_stats['user_id'].isin(['1814'])

sns.distplot(users_stats['delta_revenue_%'][mask], kde=False)
```

[313]:
```
<AxesSubplot:xlabel='delta_revenue_%'>
```



[314]:
```
users_stats.head()
```

[314]:

	user_id	role	binary_role	count post_grading_price_difference_%	\
0	100	Middle Man	non_farmer	166	
1	1000	Middle Man	non_farmer	124	
2	1004	Middle Man	non_farmer	5	
3	1011	Amprah Farmer	farmer	3	
4	1015	Middle Man	non_farmer	1	

	nanmean post_grading_price_difference_%	\
0	0.215894	
1	0.470633	
2	1.411578	
3	0.000000	
4	0.000000	

	nanmin post_grading_price_difference_%	\
0	-4.512613	
1	-0.469399	
2	0.000000	
3	0.000000	
4	0.000000	

```
   nanmedian post_grading_price_difference_%  \
0                                        0.0
1                                        0.0
2                                        0.0
3                                        0.0
4                                        0.0

   nanmax post_grading_price_difference_%  \
0                                6.635789
1                                8.913139
2                                7.057891
3                                0.000000
4                                0.000000

   nanstd post_grading_price_difference_%  counterfactual_revenue_mean  \
0                                1.661615                1.313902e+09
1                                1.612499                1.903841e+09
2                                3.156385                3.573820e+07
3                                0.000000                1.537615e+07
4                                     NaN                9.068798e+06

        revenue  transaction_count  delta_revenue  delta_revenue_%  \
0  1.311524e+09              165.0  -2.378462e+06        -0.181023
1  1.915213e+09              130.0   1.137295e+07         0.597369
2  3.636220e+07                3.0   6.240000e+05         1.746031
3  1.537615e+07                3.0   0.000000e+00         0.000000
4  9.068798e+06                1.0   0.000000e+00         0.000000

   last_upload_order last_upload_date  first_upload_order first_upload_date  \
0                153       2021-02-15                  1        2019-11-05
1                 94       2021-02-08                  1        2020-06-29
2                  4       2020-12-29                  1        2020-07-04
3                  4       2021-01-27                  1        2020-05-18
4                  1       2020-10-04                  1        2020-10-04

  first_last_upload_date_difference  average_upload_period
0                         468 days              3.078947
1                         224 days              2.408602
2                         178 days             59.333333
3                         254 days             84.666667
4                           0 days                   NaN
```

[315]:
```
users_receipt.groupby(['binary_role'])['post_grading_weight'].mean()
```

[315]:
```
binary_role
farmer        2717.199785
```

```
non_farmer    4923.359660
Name: post_grading_weight, dtype: float64
```

[316]:
```
pd.pivot_table(users_receipt,index='binary_role', values =␣
↪'post_grading_weight',
               aggfunc = [np.mean,lambda x: stats.sem(x.dropna())], margins␣
↪=True
              )
```

[316]:

	mean	<lambda>
	post_grading_weight	post_grading_weight
binary_role		
farmer	2717.199785	85.222168
non_farmer	4923.359660	41.764884
All	4624.973288	39.572575

8 Exporting File

[317]:
```
users_receipt.to_excel(f"{output_path}" + "users_receipt_final.xlsx")
```

[ ]:
```

```

Appendix D

Reproduction Code : Heterogeneity and Subgroup Analysis

Heterogeneity and Subgroup Analysis

import excel "C:\Users\AK\Dropbox (MIT)\Aufar-Joann\Analysis\Notebook 6 output\regression_table_hte_pgp.xlsx", sheet("Sheet1") firstrow clear

PGP

reg post_grading_price_treat* c_up* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, excel

reg post_grading_price_treat* c_num_local_price_available_5* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

reg post_grading_price_treat* c_s* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

reg post_grading_price_treat* c_up* v_* m_* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

reg post_grading_price_treat* c_num_local_price_available_5* v_* m_* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

reg post_grading_price_treat* c_s* v_* m_* if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

```
cvlasso post_grading_price_treat* c_up* c_num_local_price_available_5* c_s* v_* m_* if treatment_dummy==1 , seed(123)

cvlasso, lse

reg post_grading_price_treat* c_up* v_vd_BATANGGANSAL__BELIMBING v_vd_LIRIK__JAPURA v_vd_LIRIK__PASIRRINGGIT v_vd_PASIRPENYU__PETALONGAN v_vd_RAKITKULIM__PETONGGAN v_vd_RENGAT__SEKIPHILIR v_vd_SUNGAILALA__SUNGAILALA v_vd_SUNGAILALA__TANJUNGDANAU m_md_202001 m_md_202003 m_md_202101 if treatment_dummy==1 & post_grading_price_treatment_eff >-300, r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

pdslasso post_grading_price_treat* c_up* (v_* m_*) if treatment_dummy==1 , r cluster(user_id)

outreg2 using regression_hte_nb_0727, append excel

### New Buyer

logit new_buyer c_up* if treatment_dummy==1 , r cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, excel

logit new_buyer c_num_local_price_available_5* if treatment_dummy==1 , r cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, append excel

logit new_buyer c_s* if treatment_dummy==1 , r cluster(user_id)
```

```
margins, dydx(*) post

outreg2 using regression_nb_0727, append excel

logit new_buyer c_up* v_* m_* if treatment_dummy==1 , r cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, excel

logit new_buyer c_num_local_price_available_5* v_* m_* if treatment_dummy==1, r
cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, excel

logit new_buyer c_s* v_* m_* if treatment_dummy==1 , r cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, append excel

cvlassologit new_buyer c_up* c_num_local_price_available_5* c_s* v_* m_* if
treatment_dummy==1, nfold(10)

cvlassologit, lse

logit new_buyer c_num_local_price_available_5* c_s* v_vd_BATANGCENAKU__BUKITLINGKA
v_vd_BATANGCENAKU__KUALAKILAN v_vd_BATANGGANSAL__BELIMBING
v_vd_BATANGGANSAL__DANAURAMBAI v_vd_BATANGGANSAL__PANYAGUAN
v_vd_BATANGGANSAL__RINGIN v_vd_BATANGGANSAL__SEBERIDA
v_vd_BATANGGANSAL__SIAMBUL v_vd_BATANGGANSAL__SUNGAIAKAR
```

v_vd_BATANGGANSAL__TALANGLAKAT v_vd_BATANGGANSAL__USUL
v_vd_BATANGPERANAP__SELUNAK v_vd_CONCONG__SUNGAIBERAPIT v_vd_ENOK__PENGALIHAN
v_vd_KELAYANG__BONGKALMALANG v_vd_KELAYANG__DUSUNTUAPELANG
v_vd_KELAYANG__PELANGKO v_vd_KELAYANG__POLAKPISANG v_vd_KEMPAS__HARAPANTANI
v_vd_KEMPAS__KEMPASJAYA v_vd_KEMPAS__KULIMJAYA v_vd_KEMPAS__RUMBAIJAYA
v_vd_KEMUNING__KEMUNINGMUDA v_vd_KEMUNING__KEMUNINGTUA
v_vd_KUALACENAKU__KUALACENAKU v_vd_KUALACENAKU__KUALAMULYA
v_vd_KUALACENAKU__PULAUGELANG v_vd_KUALACENAKU__PULAUJUMAT
v_vd_KUALACENAKU__TELUKSUNGKAI v_vd_LIRIK__GUDANGBATU v_vd_LIRIK__JAPURA
v_vd_LIRIK__LAMBANGSARIIV v_vd_LIRIK__PASIRRINGGIT v_vd_LIRIK__PASIRSIALANGJAYA
v_vd_LIRIK__REDANGSEKO v_vd_LUBUKBATUJAYA__LUBUKBATU
v_vd_LUBUKBATUJAYA__PONDOKGEL v_vd_LUBUKBATUJAYA__RIMPIAN
v_vd_LUBUKBATUJAYA__SEIBEBERA v_vd_PANGKALANLESUNG__PANGKALAN
v_vd_PANGKALANLESUNG__SARIMULY v_vd_PASIRPENYU__AIRMOLEKI
v_vd_PASIRPENYU__CANDIREJO v_vd_PASIRPENYU__PETALONGAN
v_vd_PASIRPENYU__SERUMPUNJAYA v_vd_PERANAP__KATIPOPURA
v_vd_PERANAP__SERAIWANGI v_vd_RAKITKULIM__BUKITINDAH v_vd_RAKITKULIM__KELAYANG
v_vd_RAKITKULIM__PETONGGAN v_vd_RAKITKULIM__TALANGPERIGI
v_vd_RAKITKULIM__TALANGSELANTA v_vd_RENGATBARAT__AIRJERNIH
v_vd_RENGATBARAT__DANAUTIGA v_vd_RENGATBARAT__PEMATANGJAYA
v_vd_RENGATBARAT__PEMATANGREBA v_vd_RENGATBARAT__RANTAUBAKUNG
v_vd_RENGATBARAT__SUNGAIDAWU v_vd_RENGATBARAT__TALANGJERINJ
v_vd_RENGATBARAT__TANAHDATAR v_vd_RENGATBARAT__TANIMAKMUR
v_vd_RENGAT__KAMPUNGBESARSEBER v_vd_RENGAT__KAMPUNGKOTABESAR
v_vd_RENGAT__KUANTANBABU v_vd_RENGAT__SEKIPHILIR
v_vd_RENGAT__SUNGAIGUNTUNGHILI v_vd_RENGAT__SUNGAIRAYA
v_vd_SEBERIDA__BANDARPADANG v_vd_SEBERIDA__BUKITMERANTI
v_vd_SEBERIDA__BULUHRAMPAI v_vd_SEBERIDA__KELESA v_vd_SEBERIDA__PANGKALANKASAI
v_vd_SEBERIDA__PAYARUMBAI v_vd_SEBERIDA__PETALABUMI v_vd_SEBERIDA__SERESAM
v_vd_SEBERIDA__SIBABAT v_vd_SEBERIDA__TITIANRESAK v_vd_SUNGAILALA__KELAWAT
v_vd_SUNGAILALA__KUALALALA v_vd_SUNGAILALA__MORONG
v_vd_SUNGAILALA__PASIRBONGKAL v_vd_SUNGAILALA__PERKEBUNANSUN
v_vd_SUNGAILALA__SUNGAILALA v_vd_SUNGAILALA__TANJUNGDANAU
v_vd_TEMPULING__SUNGAISALAK v_vd_UKUI__AIREMAS v_vd_UKUI__BUKITGAJAH
v_vd_UKUI__BUKITJAYA v_vd_UKUI__LUBUKKEMBANGSARI v_vd_UKUI__UKUII
v_vd_UKUI__UKUIII m_md_201907 m_md_201908 m_md_201909 m_md_201910
m_md_201911 m_md_201912 m_md_202002 m_md_202003 m_md_202004 m_md_202005
m_md_202011 m_md_202012 m_md_202101 m_md_202102 if treatment_dummy==1,r
cluster(user_id)

margins, dydx(*) post

outreg2 using regression_nb_0727, append excel

reg new_buyer c_num_local_price_available_5* c_s* v_vd_BATANGCENAKU__BUKITLINGKA
v_vd_BATANGCENAKU__KUALAKILAN v_vd_BATANGGANSAL__BELIMBING
v_vd_BATANGGANSAL__DANAURAMBAI v_vd_BATANGGANSAL__PANYAGUAN
v_vd_BATANGGANSAL__RINGIN v_vd_BATANGGANSAL__SEBERIDA
v_vd_BATANGGANSAL__SIAMBUL v_vd_BATANGGANSAL__SUNGAIAKAR
v_vd_BATANGGANSAL__TALANGLAKAT v_vd_BATANGGANSAL__USUL
v_vd_BATANGPERANAP__SELUNAK v_vd_CONCONG__SUNGAIBERAPIT v_vd_ENOK__PENGALIHAN
v_vd_KELAYANG__BONGKALMALANG v_vd_KELAYANG__DUSUNTUAPELANG
v_vd_KELAYANG__PELANGKO v_vd_KELAYANG__POLAKPISANG v_vd_KEMPAS__HARAPANTANI
v_vd_KEMPAS__KEMPASJAYA v_vd_KEMPAS__KULIMJAYA v_vd_KEMPAS__RUMBAIJAYA
v_vd_KEMUNING__KEMUNINGMUDA v_vd_KEMUNING__KEMUNINGTUA
v_vd_KUALACENAKU__KUALACENAKU v_vd_KUALACENAKU__KUALAMULYA
v_vd_KUALACENAKU__PULAUGELANG v_vd_KUALACENAKU__PULAUJUMAT
v_vd_KUALACENAKU__TELUKSUNGKAI v_vd_LIRIK__GUDANGBATU v_vd_LIRIK__JAPURA
v_vd_LIRIK__LAMBANGSARIIV v_vd_LIRIK__PASIRRINGGIT v_vd_LIRIK__PASIRSIALANGJAYA
v_vd_LIRIK__REDANGSEKO v_vd_LUBUKBATUJAYA__LUBUKBATU
v_vd_LUBUKBATUJAYA__PONDOKGEL v_vd_LUBUKBATUJAYA__RIMPIAN
v_vd_LUBUKBATUJAYA__SEIBEBERA v_vd_PANGKALANLESUNG__PANGKALAN
v_vd_PANGKALANLESUNG__SARIMULY v_vd_PASIRPENYU__AIRMOLEKI
v_vd_PASIRPENYU__CANDIREJO v_vd_PASIRPENYU__PETALONGAN
v_vd_PASIRPENYU__SERUMPUNJAYA v_vd_PERANAP__KATIPOPURA
v_vd_PERANAP__SERAIWANGI v_vd_RAKITKULIM__BUKITINDAH v_vd_RAKITKULIM__KELAYANG
v_vd_RAKITKULIM__PETONGGAN v_vd_RAKITKULIM__TALANGPERIGI
v_vd_RAKITKULIM__TALANGSELANTA v_vd_RENGATBARAT__AIRJERNIH
v_vd_RENGATBARAT__DANAUTIGA v_vd_RENGATBARAT__PEMATANGJAYA
v_vd_RENGATBARAT__PEMATANGREBA v_vd_RENGATBARAT__RANTAUBAKUNG
v_vd_RENGATBARAT__SUNGAIDAWU v_vd_RENGATBARAT__TALANGJERINJ
v_vd_RENGATBARAT__TANAHDATAR v_vd_RENGATBARAT__TANIMAKMUR
v_vd_RENGAT__KAMPUNGBESARSEBER v_vd_RENGAT__KAMPUNGKOTABESAR
v_vd_RENGAT__KUANTANBABU v_vd_RENGAT__SEKIPHILIR
v_vd_RENGAT__SUNGAIGUNTUNGHILI v_vd_RENGAT__SUNGAIRAYA
v_vd_SEBERIDA__BANDARPADANG v_vd_SEBERIDA__BUKITMERANTI
v_vd_SEBERIDA__BULUHRAMPAI v_vd_SEBERIDA__KELESA v_vd_SEBERIDA__PANGKALANKASAI
v_vd_SEBERIDA__PAYARUMBAI v_vd_SEBERIDA__PETALABUMI v_vd_SEBERIDA__SERESAM
v_vd_SEBERIDA__SIBABAT v_vd_SEBERIDA__TITIANRESAK v_vd_SUNGAILALA__KELAWAT
v_vd_SUNGAILALA__KUALALALA v_vd_SUNGAILALA__MORONG
v_vd_SUNGAILALA__PASIRBONGKAL v_vd_SUNGAILALA__PERKEBUNANSUN
v_vd_SUNGAILALA__SUNGAILALA v_vd_SUNGAILALA__TANJUNGDANAU
v_vd_TEMPULING__SUNGAISALAK v_vd_UKUI__AIREMAS v_vd_UKUI__BUKITGAJAH
v_vd_UKUI__BUKITJAYA v_vd_UKUI__LUBUKKEMBANGSARI v_vd_UKUI__UKUII
v_vd_UKUI__UKUIII m_md_201907 m_md_201908 m_md_201909 m_md_201910
m_md_201911 m_md_201912 m_md_202002 m_md_202003 m_md_202004 m_md_202005
m_md_202011 m_md_202012 m_md_202101 m_md_202102 if treatment_dummy==1,r
cluster(user_id)

outreg2 using regression_nb_0727, append excel

pdslasso new_buyer c_num_local_price_available_5* c_s* (v_* m_*) if treatment_dummy==1,r cluster(user_id)

outreg2 using regression_nb_0727, append excel

Marginal effect Analysis

reg post_grading_price_treat* c_up* v_vd_BATANGGANSAL__BELIMBING v_vd_LIRIK__JAPURA v_vd_LIRIK__PASIRRINGGIT v_vd_PASIRPENYU__PETALONGAN v_vd_RAKITKULIM__PETONGGAN v_vd_RENGAT__SEKIPHILIR v_vd_SUNGAILALA__SUNGAILALA v_vd_SUNGAILALA__TANJUNGDANAU m_md_202001 m_md_202003 m_md_202101 if treatment_dummy==1 & post_grading_price_treatment_eff >-300, r cluster(user_id)

margins, at((median) _all c_upload_order=(1 50 100 150 200 250 300) v_vd_SUNGAILALA__SUNGAILALA=1 m_md_202101=1)

margins, at((median) _all c_upload_order=(1 50 100 150 200 250 300) v_vd_BATANGGANSAL__BELIMBING=1 m_md_202101=1)

Correction for sub_districts dummy

cvlassologit new_buyer c_up* c_num_local_price_available_5* c_s* v_* k_* m_* if treatment_dummy==1, nfold(10) seed(123)

cvlassologit, lse

logit new_buyer c_num_local_price_available_5* c_s* v_vd_BATANGGANSAL__BELIMBING v_vd_BATANGGANSAL__RINGIN v_vd_BATANGGANSAL__SEBERIDA v_vd_BATANGGANSAL__SIAMBUL v_vd_BATANGGANSAL__SUNGAIAKAR v_vd_BATANGGANSAL__TALANGLAKAT v_vd_BATANGGANSAL__USUL v_vd_BATANGPERANAP__SELUNAK v_vd_CONCONG__SUNGAIBERAPIT v_vd_ENOK__PENGALIHAN v_vd_KELAYANG__DUSUNTUAPELANG v_vd_KEMPAS__HARAPANTANI v_vd_KEMPAS__KULIMJAYA v_vd_KEMPAS__RUMBAIJAYA v_vd_KEMUNING__KEMUNINGMUDA v_vd_KERITANG__PETALONGAN v_vd_KUALACENAKU__PULAUGELANG v_vd_LIRIK__JAPURA v_vd_LIRIK__LAMBANGSARIIV v_vd_LIRIK__PASIRRINGGIT v_vd_LIRIK__PASIRSIALANGJAYA v_vd_LUBUKBATUJAYA__LUBUKBATU v_vd_LUBUKBATUJAYA__RIMPIAN

v_vd_PASIRPENYU__CANDIREJO v_vd_PASIRPENYU__PETALONGAN
v_vd_RAKITKULIM__PETONGGAN v_vd_RENGATBARAT__TALANGJERINJ
v_vd_RENGATBARAT__TANAHDATAR v_vd_RENGATBARAT__TANIMAKMUR
v_vd_RENGAT__KAMPUNGBESARSEBER v_vd_RENGAT__KAMPUNGKOTABESAR
v_vd_RENGAT__SEKIPHILIR v_vd_RENGAT__SUNGAIRAYA v_vd_SEBERIDA__BANDARPADANG
v_vd_SEBERIDA__BUKITMERANTI v_vd_SEBERIDA__KELESA v_vd_SEBERIDA__PANGKALANKASAI
v_vd_SEBERIDA__SIBABAT v_vd_SEBERIDA__TITIANRESAK v_vd_SUNGAILALA__KUALALALA
v_vd_SUNGAILALA__MORONG v_vd_SUNGAILALA__PASIRBONGKAL
v_vd_SUNGAILALA__PERKEBUNANSUN v_vd_SUNGAILALA__TANJUNGDANAU
v_vd_TEMPULING__SUNGAISALAK v_vd_UKUI__BUKITJAYA k_kd_BATANGCENAKU
k_kd_BATANGPERANAP k_kd_ENOK k_kd_KELAYANG k_kd_KEMPAS k_kd_KERITANG
k_kd_LUBUKBATUJAYA k_kd_PERANAP k_kd_RENGATBARAT k_kd_SUNGAILALA m_md_201905
m_md_201907 m_md_201908 m_md_201909 m_md_201910 m_md_201911 m_md_202003
m_md_202004 m_md_202011 m_md_202012 m_md_202101 if treatment_dummy==1,r
cluster(user_id)

margins, dydx(*) post

pdslasso new_buyer c_num_local_price_available_5* c_s* (v_* K_8 m_*) if treatment_dummy==1,r
cluster(user_id)

cvlasso post_grading_price_treat* c_up* c_num_local_price_available_5* c_s* v_* k_* m_* if
treatment_dummy==1 , seed(123)

cvlasso, lse

reg post_grading_price_treat* c_up* v_vd_BATANGGANSAL__BELIMBING v_vd_LIRIK__JAPURA
v_vd_LIRIK__PASIRRINGGIT v_vd_PASIRPENYU__PETALONGAN v_vd_RAKITKULIM__PETONGGAN
v_vd_RENGAT__SEKIPHILIR v_vd_SUNGAILALA__SUNGAILALA
v_vd_SUNGAILALA__TANJUNGDANAU k_kd_RENGAT m_md_202001 m_md_202003 m_md_202101
if treatment_dummy==1, r cluster(user_id)

pdslasso post_grading_price_treat* c_up* (v_* k_* m_*) if treatment_dummy==1 , r
cluster(user_id)

Appendix E

Reproduction Code : Notebook 8

Subgroup Analysis

Notebook 8 Subgroup Analysis

August 29, 2021

1 Importing Library

[1]:
```
import pandas as pd
import numpy as np
import scipy
from numpy import nan
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
#sns.set_theme(color_codes=True)
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
#sns.set_theme(color_codes=True)
```

2 Importing data

[3]:
```
raw_data = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Raw Data/'

notebook_1_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 1␣
 ↪output/'

notebook_2_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 2␣
 ↪output/'

notebook_3_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 3␣
 ↪output/'

notebook_4_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 4␣
 ↪output/'

notebook_5_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 5␣
 ↪output/'
```

```
hte_analysis_output = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/HTE␣
↪Analysis Output/'

output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 6 output/
↪'

fig_output_path = 'C:/Users/AK/Dropbox (MIT)/Aufar-Joann/Analysis/Notebook 6␣
↪output/Figures/'
```

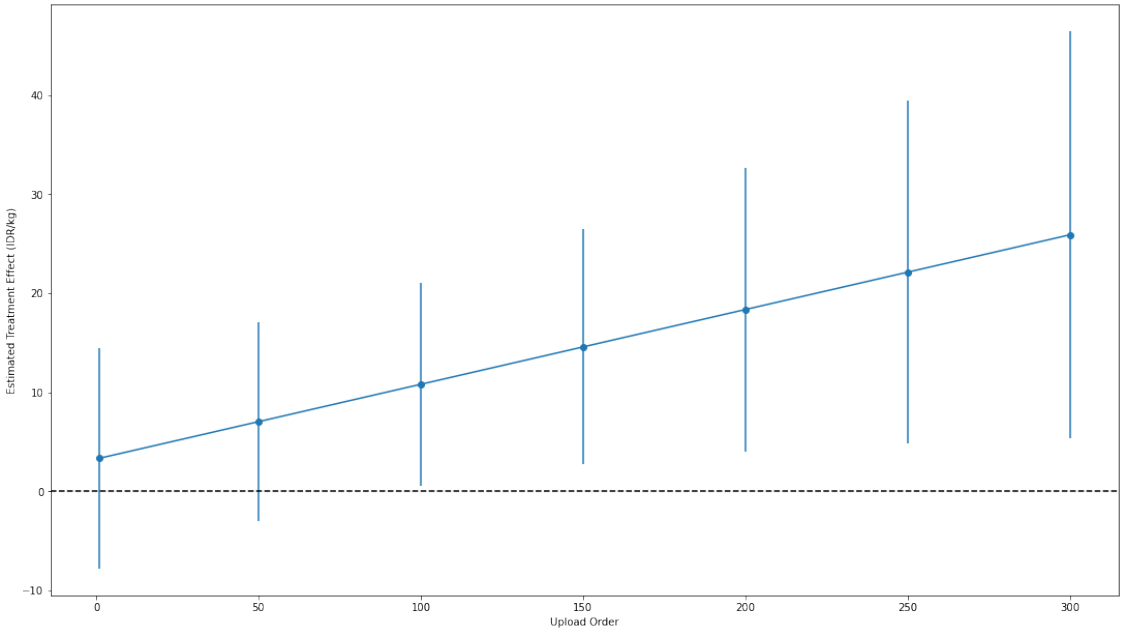
2.1 Import Margins Analysis

[4]:
```
margins_analysis= pd.read_csv(f"{hte_analysis_output}" +␣
↪"margins_analysis_final.csv")
```

[5]:
```
margins_analysis['error'] = margins_analysis['upper_95'] -␣
↪margins_analysis['lower_95']
```

3 Subgroup Analysis

[6]:
```
fig, ax = plt.subplots(figsize=(18.5, 10.5))

mask = ((margins_analysis['model'] =="Linear Model")
        & (margins_analysis['village'] =="BATANG GANSAL__BELIMBING")
       )

x = margins_analysis['c_upload_order'][mask]
y = margins_analysis['tau.hat'][mask]
error  =margins_analysis['error'][mask]/2
ax.axhline(y=0,linestyle ='--', color='k')
ax.errorbar(x, y, yerr=error, fmt='-o')
#ax.set_title('Non Linear Model')
ax.set_xlabel("Upload Order")
ax.set_ylabel("Estimated Treatment Effect (IDR/kg)")

plt.savefig(f"{fig_output_path}" +"subgroup_analysis_linear.png")
```


Estimated Treatment Effect (IDR/kg)
40
30
20
10
0
−10
0
50
100
150
200
250
300
Upload Order