Understanding Learner Engagement and the Effect of Course Structure in Massive Open Online Courses

by Christine M. Vonder Haar

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

February 2020

© Massachusetts Institute of Technology 2020. All rights reserved.

Author:

________________________
Department of Electrical Engineering and Computer Science
January 29, 2020

Certified by:

________________________
Ana Bell, EECS Principal Lecturer, Thesis Supervisor
January 29, 2020

Accepted by:

________________________
Katrina LaCurts, Chair, Master of Engineering Thesis Committee
In this thesis, we analyze learner performance in two edX programming courses. We look at many different types of learners, such as learners who have taken both intro and advanced courses, learners who opt to pay for certification, learners who take the experimental self-paced course, learners who eventually become community teaching assistants, and learners who take the course after the implementation of gating. In particular, we focus on repeat learners, or learners who have taken the course multiple times. When courses undergo a change from semester to semester, for example changing the pacing of the course or making certain content only available to paid users, it can be very useful to look at learners who were in the course before and after this change. Our goal is to gain a baseline understanding of how different factors affect learner behavior and how a few changes that edX has made to courses affect learner performance. With the best understanding of how learners interact with and complete courses, edX instructors will be able to provide the best possible online education experience for their learners.
## Contents

**List of Figures**

4

**List of Tables**

4

### 1 Introduction

1.1 Background ............................. 5
1.2 Related Work ............................ 5
1.3 Motivation .............................. 6
1.4 Courses Summary ........................ 8

### 2 Data

2.1 Data Cleaning .......................... 10
2.2 Data Sets ............................... 12
2.3 Data Parsing ............................ 13
2.4 Data Visualization ..................... 14

### 3 Learner Analysis

3.1 Factors Affecting Learner Performance ........................................ 17
3.2 Repeat learners .......................... 20
   3.2.1 Two-time repeat learners .......... 22
   3.2.2 Repeat cross-referencers .......... 24
3.3 Dual-course learners ................... 25
3.4 Community TAs ........................ 26
3.5 Verified and Certified Learners .................. 27

### 4 Course Analysis

4.1 Basics .................................. 29
4.2 Self-paced vs. Instructor-led .................. 31
4.3 Intro vs. Advanced ...................... 34

### 5 Monetary Analysis

5.1 edX Promos .............................. 38
5.2 Increase in Certification Fee ................ 39
5.3 Time- and Feature-Gating .................. 40

### 6 Conclusions

6.1 Further Work ........................... 42
6.2 Final Thoughts .......................... 43

### 7 References

44

**Glossary**

45

**Appendix A Python Scripts**

47
List of Figures

1. Courses Timeline ............................................... 8
2. Structure of data ............................................... 13
3. Data Visualization ............................................... 15
4. Comparing MOOCs with the dashboard ................................ 16
5. Pset vs. Exam Grades ........................................... 17
6. Engagement Factors By Final Grade ................................ 18
7. Completion rate vs. learner enrollment date .......................... 19
8. Repeat learner engagement ........................................ 21
9. Two-time repeat learners ......................................... 23
10. Self-paced vs. instructor-led completion rates ...................... 33
11. 1x vs. 2x Age Distribution ..................................... 35
12. 1x vs. 2x Grades .................................................. 36
13. Effect of promos on registration numbers .......................... 38
14. Assignment completion rate of unverified learners before and after gating .................. 41
15. Effects of gating on completion and verified rates .................. 42

List of Tables

1. Example of assignment schedule ................................... 9
2. Comparing dual- and single-course learners ......................... 26
3. Verified vs. non-verified engagement ................................ 28
4. Course run averages ............................................... 30
5. Learners who pass for self-paced vs. instructor-led .................. 32
1 Introduction

1.1 Background

The internet has drastically changed the way the average person can learn new subjects and skills. Specifically, Massive Open Online Courses (MOOCs) provide college-level content to anyone willing to enroll in a course. Many of these courses are provided free of charge, with the option to become certified for a nominal fee. Some platforms even offer college credit for completing certain online courses. One platform, edX, was created through a joint effort between Harvard and MIT in 2012 to provide MOOCs to the public [1]. In 2017, edX provided more than 1800 courses to more than 14 million learners [2]. In this thesis, we will be analyzing the performance of learners in two of these courses.

MIT alone has dozens of introductory and advanced courses available on edX. We will be focusing on the edX versions of 6.0001 (Introduction to Computer Science Programming in Python) and 6.0002 (Introduction to Computational Thinking and Data Science). In the rest of this paper, we will be referring to these courses as 1x and 2x, respectively. Computation is becoming an integral part of education, especially at college-level. These courses are popular because they teach not only computer science and computation ideas, but also general problem solving skills. They frame the world around computation, and show learners that they can think about problems they see in everyday life in the context of computation. The skills learners gain in these courses are useful far beyond the computer science field alone, which is part of the reason they are so popular.

1.2 Related Work

This thesis focuses on breadth, i.e. analyzing many different aspects of two MOOCs such as course pacing, learner types, monetary analysis, and course difficulty. Others have written theses on specific phenomenon within edX courses. Wang explores how grade feedback throughout the semester affects learner performance and involvement [3]. Bajwa investigates
learner trajectories based on features of individual problems as well as information about the student and their submissions [4]. Northcutt, Ho, and Chuang identify and detect cheaters in MOOCs who use multiple accounts to obtain correct answers without doing the work [5]. These theses and papers are recommended for the interested reader to learn more about these specifics area of research.

1.3 Motivation

Understanding how learners interact with a MOOC is essential in recognizing which factors affect learner performance and how instructors can tailor a MOOC to set the average learner up for success. Because MOOCs lack a component of human interaction from the traditional student/teacher learning approach, we must rely on extensive data collected during these courses to discern how users are learning with MOOCs.

Below are some questions that drove much of our investigation of the data. This is by no means a complete list, rather questions with important answers we believe we can glean from the data.

- What factors (engagement in the course, test scores, problem set scores, certification status, demographic information, etc.) demonstrate a correlation in a learner’s success in the course?

- How do repeat learners (learners who enroll in the same course multiple times) interact with the course differently?
  - Do repeat learners tend to enroll in subsequent semesters? Or do they take a break and come back to the course after a while? Is there a difference in performance between these two subgroups of repeat learners?

- How do learners who sign up for the certification and those who enroll just to learn interact with the course differently?
• How does the introductory course (1x) differ from the advanced course (2x) in engagement, retention, popularity, learner performance, and other student experience metrics?

• How do administrative or structural changes edX makes in courses affect learner engagement?

  – How do the self-paced runs of the course differ from the instructor-led runs of the course in engagement, retention, popularity, learner performance, etc?

  – How do verified and non-verified learners interact with the course before and after content restriction is introduced in courses?

• How can we partition the vast pool of learners into smaller sub groups?

  – For example, some learners don’t do anything within the course besides the graded problems and the tests. Some learners look back at previous runs of the course (potentially to help them in the current run of the course). Some learners eventually become community TAs. How do these groups perform differently?

• Would it make sense to change the structure of the course for potential benefit of the average learner?

  – For example, would requiring learners to watch a certain percentage of videos lead to better grades in the course overall? What about mandatory involvement in the forum?

• At what point do students who don’t complete the course quit?

• Can this data analysis be extrapolated to other MIT introductory courses on edX? Do we see similar results across courses, or will the results vary based on course subject and content?
1.4 Courses Summary

1x is considered an introductory course, with the assumption that the learners have never been exposed to any programming before, while 2x is considered a more advanced course and does require some prior knowledge of Python before taking the course. Figure 1 shows how frequently and at what times 1x and 2x have been offered on edX since 2013. 1x has been offered 16 times and 2x has been offered 11 times as of Fall 2019. Usually, 1x is offered three times a year with start dates in mid January, early June, and late August; 2x is offered twice a year in early March and mid October. Courses can be offered up to three times per year, designated by “1T,” “2T,” or “3T.” We usually call a course offered in a specific timeframe a “semester” or “run” of the course. Course enrollment can also vary a lot year to year.

Learners can enroll as many times as they want for a course. Sometimes, learners don’t

Figure 1: The timeline of each course over the past 6 years. The lighter boxes indicate when 1x was offered and the darker boxes indicate when 2x was offered. The instructor-led versions of the course are open about 2-3 months while the self-paced version was open for an entire year (see Section 4.2).
even view a course they register for. An even smaller number explore a course, which means they visited at least half of the chapters in a course. While most learners opt to take the course for free, a learner can become verified by paying a certification fee. This means that if they finish the course with a passing grade, they will be able to download and distribute a certification of completion.

Many factors can affect learner enrollment. A course is often more popular the first few times it is ever offered. In late 2016, both courses were revamped and highly advertised when the switch to Python 3 was made, which resulted in higher enrollment in subsequent semesters. edX also often promotes 1x and 2x in emails to learners, which also affects enrollment and learner engagement (see Section 5.1). Another important change in the way courses are run occurred in spring of 2019, when edX rolled out time- and feature-gating for 1x and 2x. This change prevents non-verified learners from visiting the course content after the course ends and from viewing the exams in the course (see Section 5.3).

In each course, learners complete a variety of assignments in order to complete the course. The assignments that contribute to a learner’s overall grade are finger exercises, problem sets,

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Start Date</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem set 1</td>
<td>January 22</td>
<td>February 7</td>
</tr>
<tr>
<td>Problem set 2</td>
<td>January 30</td>
<td>February 14</td>
</tr>
<tr>
<td>Problem set 3</td>
<td>February 10</td>
<td>February 21</td>
</tr>
<tr>
<td>Problem set 4</td>
<td>February 20</td>
<td>March 7</td>
</tr>
<tr>
<td>Midterm exam</td>
<td>February 22</td>
<td>February 26</td>
</tr>
<tr>
<td>Problem set 5</td>
<td>February 27</td>
<td>March 14</td>
</tr>
<tr>
<td>Problem set 6</td>
<td>March 10</td>
<td>March 21</td>
</tr>
<tr>
<td>Final exam</td>
<td>March 22</td>
<td>March 26</td>
</tr>
</tbody>
</table>

Table 1: The table shows the deadlines of all assignments in the spring 2019 run of 1x. The timeline of events for 2x is organized in a similar way. The time frames of assignments usually overlap; two problem sets are usually active at one time. If learners don’t complete an assignment within the specified time frame, they won’t receive credit.
a midterm, and a final. *Finger exercises* are short problems, with solutions of < 20 lines of code, that are linked with each video lecture and learners can actually see the answers to these problems. *Problem sets*, also known as psets, are longer assignments stretched over about 2 weeks in which learners tackle a larger problem. Each assignment is weighted differently – the midterm and final are each worth 25%, the problem sets are together worth 40%, and the finger exercises are worth 10%. A learner must earn an overall grade 55% or above in order to complete the course. An example schedule of a spring semester of 1x is shown in Table 1.

In each run, there is an online *forum* where students can ask questions and interact with each other and the course staff. Students can post to the forum if they have questions about any assignments or about any of the course content in general. Some learners who liked the course even come back and become community TAs, who are volunteers that monitor the forum and answer students’ questions throughout the semester.

2 Data

Data from the 1x and 2x courses from 2013 to 2019 is being analyzed. The data edX collects is varied, from the course start and end dates to how much the course was promoted to how many times a learner attempted each problem to demographic info and much more.

2.1 Data Cleaning

Tens of thousands of learners enroll in both 1x and 2x every run of the course. Both courses run 2 or 3 times a year since 2013, which results in a very significant amount of data to be analyzed. For each run of a course, there is over a GB of data to consider. This is a great opportunity to get some real results and insights about the courses. On the downside, 

---

1 The lowest problem set score is dropped when calculating the overall pset average.
the data edX provides is not always in the best format. Data cleaning is a very important preliminary step that will ensure our results are as accurate as possible.

For obvious reasons, we remove all the instructors and community TAs from the data set before analysis. We will be looking more at community teaching assistants (CTAs), specifically CTAs who used to be students, in Section 3.4.

The next thing we did was remove all the learners who registered for the course, but never actually viewed it. Because these courses are provided free of charge, thousands of users enroll for them without ever logging onto the course. Historically, around 40-50% of the users who register for a course never log onto it. We usually only want to consider active learners, or learners who at least view the course once. These inactive users don’t provide us any valuable information or insight beyond enrollment numbers, so we usually remove them from our data set (unless we are looking specifically at enrollment numbers).

Another thing we looked at was repeat cross-referencers in the course. We defined a repeat cross-referencer as a learner who accessed at least two previous runs of the course during the run time of a later run of the course. Because the course doesn’t change much from semester to semester, we speculate that learners who repeatedly access previous courses may be looking at answers from previous runs and not actually doing the work themselves. While we cannot definitely prove these users are cheating, we have a reasonable suspicion that they are not doing the work anew each semester and therefore don’t consider them in the more general analytics we perform. As mentioned above, we put them in their own category and analyze them in Section 3.2.2 to see if there are any perceptible differences in the performance of those we think are repeat cross-referencers and those who aren’t.

Finally, we remove learners who don’t interact with the course beyond completing the graded problems. This includes learners who complete the course and never watch videos or participate in the forum. We speculate there are at least some learners who actually already know how to code and are going through the course specifically for the certificate or completion, instead of to actually learn Python. While these learners are still valid, we don’t
include them in our general analysis simply because they are not actually learning from and interacting with the course. There are less than 500 of these noninteractive learners over all runs of each course, so removing them does not affect results in a significant way.

Even with removing so many learners for all the reasons stated above, we still have plenty of data to analyze. Some analysis is done before removing the learners who never view, because sometimes it is useful to look at percentages based on total number of learners registered. Before removing these never viewed learners, we had about 1.38 million learners to analyze over all runs of 1x and over 200,000 learners in 2x. After removing these learners who never view the course and only considering active learners, which is the data we use in the majority of analysis, we were left with around 857,000 learners in 1x and 122,000 learners in 2x.

2.2 Data Sets

For each run of each course, edX provides multiple data sets of everything that is recorded for each user—down to specific clicks and actions of each user. Three of the datasets are person_course, grade_report, and forum_person. The person_course data set contains aggregate data about each learner during a run of the course, including whether or not a learner viewed, explored (visited over half of the chapters), and completed the course, the number of days the learner was active on the course, how many video and forum actions they had, and many more pieces of data. The grade_report data set contains the overall grades for each assignment, problem set, and assessment in the course. The forum_person data set is all the forum events for each person in a course. From this file, we get the total number of posts a learner has written, read, and followed throughout the course.

These are just three of ten or more data sets available for each run of the course. However, they are by far the most useful data sets. Many other data sets provided by edX are too fine-grained (for example, one data set includes every submission for each problem for every user) or not fine-grained enough (for example, one data set includes the overall completion
and exploration rates for every learner in the course) or do not contain useful data for our purposes. The data sets we have chosen are the perfect middle-ground in terms of data granularity: descriptive enough that we can partition and analyze learners in a meaningful way, but not so descriptive that we have to deal with parsing individual user events.

2.3 Data Parsing

All of this data is read and organized in our Python scripts. The data is organized in two ways: by semester and by student. While it’s not necessary to organize all the data in this way, it makes the analysis of both groups of learners and semesters of a course much easier. The general structure of the data is laid out in Figure 2. We have two super classes: BigGroup with two children Course and Cohort and SmallGroup with two children Semester and Student. Both Semester and Student contain multiple instances of Entry, which represents one student during one semester and contains all the data, such as username,
grades, number of videos watched, total number of forum events, and more. A Course contains Semester objects that represents all the runs of either 1x or 2x. A Cohort contains Student objects that represents a group of students including all the data from each time a student took the course. Usually a Cohort contains data from either 1x or 2x, but data from both courses can be considered together if we are looking at students who took both.

All analysis is performed within these classes. We can get the average completion rate of an entire Course or Cohort, or just a Semester or Student individually. Once all the data is loaded, it is passed to various scripts to perform analysis. In general, there’s one script for each section, e.g. one script analyzing the changes in courses and another script analyzing learner subgroups. This isn’t completely necessary, but just helps to compartmentalize sections because there’s over one thousand lines of code. For more information about the scripts used to perform all the data analysis, see Appendix A.

We have such an elaborate structure for the data that holds everything we could possibly need because it takes a while to load all the data, so we only want to do it once. We load the data in a script main.py, which reloads other analysis scripts when prompted. As long as we don’t change how the data is loaded or the class instances, we are able to perform new analysis without having to completely reload the data each time. Having everything held in a single instance such as Course or Cohort makes it easy to pass around all the data to the other scripts and perform analysis quickly.

2.4 Data Visualization

A semi-interactive data visualization, including some of the graphs presented in this paper, can be found at http://mooc-viz.mit.edu/. See Figure 3 for an example of what the dashboard looks like. This dashboard will help edX teachers of all courses more easily visualize and interact with his/her data from semester to semester. All the code for the visualization, as well as a README explaining the inputs required for each graph, is available in a GitHub repository at https://github.com/vonderhaar/MEng-thesis/tree/master/
Figure 3: The data visualization dashboard for edX teachers. This dashboard is hosted at http://mooc-viz.mit.edu/

visualization. This tool could provide a simple framework in which to compare and contrast all edX courses, and can also be customized to show additional graphs that are particular to a certain MOOC.

A researcher looking at the equivalent edX biology MOOC data was able to send us some of their aggregate data. Figure 4 shows a side by side of the learner engagement of 1x and the introductory biology MOOC. We see a similar trend of viewed/explored/completed in the two courses – around 60-70% of learners will view a course, usually 5-15% will explore it, and less than 5% complete it in both courses. Any analyst looking at edX data for their specific course will either already have this data or will be able to generate it fairly easily. This simple comparison is just a proof of concept that the dashboard could be extremely helpful and easy to use for many different edX MOOCs.
Programming MOOC engagement

Biology MOOC engagement

Figure 4: An example of a graph on the visualization dashboard for two different MOOCs. We see similar trends in learners viewing, exploring, and completing the course, even though the courses are completely different subjects and structures.
3 Learner Analysis

3.1 Factors Affecting Learner Performance

Many factors affect how well learners perform in a course in any given semester. Some of these are more nuanced, such as whether the course was self-paced or instructor-led (Section 4.2) or if the learner has already taken the course before (Section 3.2). But first, we will investigate basic features of a learner that will likely affect their grades and performance in these MOOCs.

Figure 5 shows a scatter plot of the pset and exam grades of a sample of learners who completed each course. There is a clear diagonal boundary; the learners who would be represented in the lower left corner of the graph are learners who did not pass the course. Of those who passed the course, a majority have a high average (greater than 80%) on their

\[ \text{Figure 5: The graphs show the pset average and exam average for a subset of learners who completed the course. We see similar trends in both courses: learners who complete are likely to have high scores on psets and there are very few learners who have low pset scores who do well on exams.} \]
Figure 6: The graphs above show how many active days, total videos watched, and total events on the class forum binned by learners with increasing final grades, with 1x on the left and 2x on the right. In general, the total number of events in all three categories increases as a learner’s grades increases.

psets. Some learners who have a high pset average still don’t do well on tests, but the higher their pset average the more likely they are to do well on the exams. In contrast, very few learners do poorly on their psets but score high grades on the exams. Though perhaps an obvious conclusion, learners should be encouraged to do their psets (and do them well) in order to score high on exams and in the class overall.

The number of videos watched, the number of events on the class forum, and the number of active days in the course are useful metrics of learner engagement in the course. In Figure 6, learners are grouped by their final grade in the course. The average number of videos watched, the average number of forum events (including posts, comments, followings, and
views), and the average number of days active is plotted for each bin. The number of videos watched and the number of active days strictly increases with an increase in final grade. The number of forum events generally increases, but not strictly. Being active on the forum could be a result of not understanding the course material and therefore asking questions or searching in existing posts for help, but it could also mean being proactive about confusing in-course content or even answering other questions for students. Going deeper into the forum data is out of the scope of this thesis, but at first look there is definitely positive correlation between forum activity and learner performance.

Some learners join a course after the course has actually started, as completing all the psets and finger exercises is not necessary to complete and pass the course. Learners can enroll many days before or after the course start date. After the midterm happens, learners have likely lost too many points to earn a passing grade in the course, but edX will still

---

3We will be excluding the self-paced version of the course from this analysis, as the course was open an entire year and there were no due dates, so learners could enter at any point between the course start and end dates and complete all the necessary material.

Figure 7: The above graph shows completion rates vs. learner enrollment dates. Positive x values mean the learner enrolled for the course before it started; negative x values mean the learner enrolled after the course started and started late.
allow them to enroll. Figure 7 explores the relationship between learner enrollment date (relative to course start date) and completion rate. The graphs take on a S-curve shape: at some point, starting too late will drastically hurt a learner from completing the course but enrolling very early doesn’t give too much of an advantage. The most drastic change in completion rate happens the 20 days or so after the course begins. After 20 days, usually two psets have been assigned and due. This indicates learners can’t catch up after missing the first three weeks of work in course.

Given these engagement factors that clearly indicate the success of students, it would be interesting for edX to potentially require a certain level of engagement in videos and the forum in order for learners to pass the course. In theory, requiring learners to spend a certain amount of time with course content would improve overall performance. However, if a threshold was required in order to pass the course, there could be an adverse effect in that learners will not want to put in this extra effort and many learners who could have completed the course without the extra required work won’t even attempt to complete it. It may be better to instead relay these findings to learners in order to encourage their engagement rather than requiring it.

3.2 Repeat learners

A repeat learner is defined as someone who has registered for the same course two times or more (e.g. a learner who takes 1x once and 2x once does not count). Around 20% of learners in any given semester of a course are repeat learners, as we see in Table 4 in Section 4.1. Our hypothesis is that repeat learners will engage more in the course and complete the course at a higher rate than the average learner registering for the course for the first time.

As we can see in Figure 8, repeat learners do seem to be more engaged in a course in subsequent semesters. The plots show the percent of learners who ever viewed, explored, and
completed the course (in any semester). All three categories steadily increase as learners register for more and more semesters of the same course. This makes intuitive sense: if a learner isn’t satisfied with their performance in a previous run of the course, they come back to a subsequent run of the course with more intention and a better expectation of what needs to be done to do well. We also notice this repeat learner phenomenon affects 1x runs more than 2x runs. We can rationalize this in two ways: (1) learners who register for the advanced course have likely taken other MOOCs before and therefore have more intention and a better expectation even if its their first time in this specific course and (2) the content in the introductory course is easier to understand, so learners have a better

---

4 This data has been parsed in the ways described in Section 2.1 with the exception of removing students who never viewed the course. After these graphs were generated, students who never viewed the course were removed for the remaining analysis.
chance at picking it up successfully in subsequent runs when they’ve already been exposed to some of the material. It’s also important to note that fewer people take the course many, many times. For example, only one person has registered for 1x all 16 times whereas 4188 learners have registered for exactly 5 semesters. The less learners in a bin, the less significant the result is.

Overall, the completion rate of a one-time learner is 3.5% while repeat learners eventually complete 1x at a rate of 9.7%. In 2x, the first time learner completion rate is 5.7% and the repeat learner completion rate is 11.0%. Repeat learners are at least twice as likely to eventually pass the course than one-time learners.

3.2.1 Two-time repeat learners

This section will focus on learners who took the course exactly twice. The majority of repeat learners take a course exactly twice: 72% in 1x and 75% in 2x of the total number of repeat learners were two-time repeat learners. Around 1% of two-time repeat learners actually completed the course the first time they took it, but still registered for another semester. This is usually because some learners register for the free version and complete it before paying for certification in a subsequent semester. These students were removed for the following analysis.

Within two-time repeat learners, most retake the course very soon after they take a course for the first time. In 1x, 37% of learners retook it the subsequent run, 19% retook it after skipping one run in between, and 44% retook it after two or more runs of the course had passed. In 2x, 52% learners retook it the subsequent run, 21% retook it after skipping one run in between, and 27% retook it after two or more runs of the course had passed. The top graphs of Figure 9 show that as the number of semesters in between runs increases, the number of learners in that bin decreases exponentially.

The bottom graphs of Figure 9 shows the breakdown of learner completion based on when the repeat learner registered for their second run. As we can see, learners who take
the second run soon after their initial run are generally more successful than those who wait a while between runs. This fact could be useful in encouraging learners who don’t succeed in their first run of the course: it’s better to take it again sooner rather than later. Another interesting phenomenon we see in both the 1x and 2x graphs is an uptick in completion after 8 and 6 runs, respectively. This could partially be due to the smaller size of the group that takes their second run of the course so much later, but it could also indicate learners coming back to the course with a stronger intention to finish it. It’s plausible that many learners register for the subsequent run of a course almost absent-mindedly, but after a longer period

Two-time repeat learners’ second semester

1x 2x

Figure 9: The top graphs show the total number of two-time learners vs. how many semesters occurred between their two runs of 1x (left) and 2x (right). The bottom graphs show the completion rate of two-time learners in their second semester again vs. how many semesters occurred between their runs.
of time has passed, some learners may come back to the course with a renewed motivation to actually finish it.

### 3.2.2 Repeat cross-referencers

Another type of repeat learner is a repeat cross-referencer. We define a repeat cross-referencer as someone who has taken the same course for multiple semesters and has accessed at least two previous courses at the same time they were taking another, later course. In 1x, over all semesters, 2413 learners fall into this category and in 2x, only 175 learners. Given hundreds of thousands of unique learners have taken 1x and 2x, a very small percentage of learners fall into this potential cheater category. These learners were removed, as described in Section 2.1, but we still want to analyze their performance and see if there’s more reasonable evidence to believe they might be gaming the system.

We looked at completion rates, both overall and in the semesters these learners were flagged for cross-referencing. Overall, cross-referencers in 1x eventually complete the course (meaning they pass at least one semester) at a rate of 33% and cross-referencers in 2x eventually complete at a rate of 32%. These completion rates are suspiciously high when compared with the average repeat learner (9.7% for 1x and 11% for 2x). Breaking down which semester these cross-referencers completed in, we found that 61.4% cross-referencers who eventually completed 1x completed in a semester in which they were flagged for cross-referencing. In 2x, the completion rate in cross-referencing semesters is 72.2%.

Clearly, when learners look back at old courses, they are more likely to complete the course. This wouldn’t be as much of an issue if they were only looking at their own old psets and other work. However, there are often discussions on the forum about pset and exam solutions, and these cross-referencers could be utilizing those resources that are not their own work. Without more fine-grained data about what exactly these learners are accessing when referring to old courses, we won’t know for sure. However, given the trends we saw above and how strict our filtering method is (learners must look back in at least two previous courses
to be flagged), we can be reasonably sure about removing them from the rest of analysis.

Catching repeat cross-referencers in this way will be less viable in future semesters. Due to time-gating (see Section 5.3), only verified learners will have access to course content after the course has ended. While it’s still possible for verified learners to potentially cross-reference, the overall number of repeat cross-referencers will be even lower than it was before this change in course structure.

3.3 Dual-course learners

_Dual-course learners_ are learners who take at least one semester of 1x and one semester of 2x. Only 9% of learners who ever took 1x also took 2x. On the other hand, 62% of learners who ever took 2x took 1x at some point. This trend is what we expect. Many 1x learners don’t go on to take more programming classes, but because 2x assumes some prior knowledge of Python and 1x and 2x are offered on the same platform, the majority of 2x learners took 1x.

Table 2 breaks down different stats for dual-course and single-course learners in 1x and 2x. Dual-course learners have a much higher overall completion rate of semesters at 18.7%. Breaking that down further, 23.5% of dual-course learners complete 1x and 12.2% of dual-course learners complete 2x. A learner in 1x will only go on to 2x if they did well, hence the much higher overall completion rate. However, once some of these dual-course learners reach the higher level course, they might realize the coursework is much harder or not as doable as the intro course. This accounts for the decrease in completion rate as well as the decrease in number of active days and overall grade in dual-course learners from 1x to 2x.

Clearly, 2x learners who took 1x at some point have an advantage. However, learners who actually complete 1x have an even greater advantage in 2x. As we saw in Table 2, 2x learners are about four times as likely to complete 2x if they took 1x. This likelihood doubles when 2x learners actually complete 1x: 25.7% of 2x learners who ever completed 1x also complete 2x.
Table 2: The table compares learners who have taken – not necessarily completed – both 1x and 2x (dual) to learners who have only taken one or the other (single).

<table>
<thead>
<tr>
<th></th>
<th>1x dual</th>
<th>1x single</th>
<th>2x dual</th>
<th>2x single</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average completion rate</td>
<td>23.5%</td>
<td>3.3%</td>
<td>12.2%</td>
<td>3.1%</td>
</tr>
<tr>
<td>Ever complete rate</td>
<td>38.5%</td>
<td>3.8%</td>
<td>15.2%</td>
<td>3.2%</td>
</tr>
<tr>
<td>Active days</td>
<td>23.5</td>
<td>9.1</td>
<td>14.3</td>
<td>5.7</td>
</tr>
<tr>
<td>Grade</td>
<td>87.3</td>
<td>81.3</td>
<td>84.6</td>
<td>84.6</td>
</tr>
</tbody>
</table>

3.4 Community TAs

A community TA (CTA) is a volunteer teaching assistant who answers questions on the community forum. Answering these questions usually involves explaining general concepts to learners, help with debugging problem sets, and explaining quiz questions. CTAs are usually former students who enjoyed and did well in the course. CTAs come back to help learners through the course and to solidify their understanding of the course by answering questions. As a reminder, forum event data includes total number posts written, read, and followed by a learner.

There are an average of 9.5 CTAs for each semester of 1x and 5.3 CTAs for each semester of 2x. Of the 92 total CTAs for 1x over the 16 semesters, 69 of them took the course at least once. The average grade of CTAs who completed the course was 87.1%, which is a few points higher than the average grade of 84.5% in 1x. Of the 44 total CTAs for 2x over the 11 semesters, 26 of them took the course at least once. The average grade of CTAs who completed the course was 90.1%, which is also higher than the average grade of 84.8% in 2x. Learners who eventually became CTAs were also much more engaged in the forum than the average learner – in 1x, these soon-to-be CTAs averaged 6674 forum events and in 2x, they averaged 8293 forum events.

Often, CTAs are CTAs for more than one semester. In 1x, 19 CTAs were CTAs for at least two 1x semesters; in 2x, 6 CTAs participated in at least two 2x semesters. Overall, 39
CTAs were CTAs for at least one semester of 1x and one semester of 2x. Repeat CTAs are slightly more engaged and committed to their roles. The average number of forum events for single-semester CTAs is 22,988 while the average number of forum events for repeat CTAs is 24,646. This is likely because after being a CTA for multiple semesters, repeat CTAs know how to answer most questions quickly.

3.5 Verified and Certified Learners

A subset of users in each run of a course sign up for certification. They pay a nominal fee (which changed in fall 2018, see Section 5.2) and, if they complete and pass the course, they receive a certificate that they can use to prove their completion to schools, employers, etc. **Verified learners** are those who pay for the certification fee during the course. There is a cutoff date for signing up to be verified, which is usually around the time of the midterm exam. **Certified learners** are verified learners who completed the course with a passing grade.

Since verification has been offered in 2015, 37,790 learners in 1x and 5624 learners in 2x signed up for the certification. That comes out to about 3.1% of learners in 1x and 3.3% of learners in 2x. Only 4.0% of learners who pay the certification fee never view the course. In comparison, 40% of non-verified learners who enroll in the course never view it.

Verified learners are also more likely to do well and actually complete the course than their non-verified counterparts. 54.2% of verified learners in 1x and 2x complete the course, while only 3.4% of non-verified learners complete. Verified learners who complete the course earn an average grade of 86.6% while non-verified learners who complete the course earn an average grade of 82.8%. Table 3 shows verified learners are also more active in the course. Not only are verified learners completing the course at a much higher rate, they are also demonstrating a better understanding of the course material.

Perhaps, though, verified learners are just more motivated in general than non-verified learners? We decided to look at learners who took a course at least one time unverified and at least one time verified. These learners who eventually do become verified actually
<table>
<thead>
<tr>
<th></th>
<th>grade</th>
<th>active days</th>
<th># videos</th>
<th># forum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verified</td>
<td>86.6%</td>
<td>38.7</td>
<td>41.6</td>
<td>728</td>
</tr>
<tr>
<td>Non-verified</td>
<td>82.8%</td>
<td>9.7</td>
<td>8.8</td>
<td>317</td>
</tr>
</tbody>
</table>

Table 3: This table compares the overall engagement of verified and non-verified learners. As we can see, verified learners score better, are active on the course more, watch more videos, and interact with the forum more than their non-verified counterparts.

are slightly more motivated than the average learner: 5% of learners who eventually become verified completed the course in their unverified semester and 66% completed in their verified semester. Both these completion rates are higher than the average, but there is still a huge difference in the same learners during their verified and non-verified semesters. We might expect the completion rate for these learners in their unverified semester to be 0% — but as we mentioned in Section 3.2 some learners complete a course first as a non-verified user and then again in another semester where they pay for certification, to ensure they don’t waste the money if they cannot complete the course.

While $50-$75 could be a hefty fee to pay for the course for some learners, it’s likely not a huge payout for the majority of learners. The reason verified learners do so much better in these MOOCs than non-verified learners is likely the same reason the no-show rate of free events is so much higher than events that charge a small fee [6]: when people put money towards something, even if the amount is minimal, they are more likely to follow through. Given this insight, it might make sense for edX to actually lower the certification fee to an amount in the $30 range. This could encourage a larger percentage of learners to not only become verified, but also complete the course because of the nominal fee they are investing in the course and themselves.
4 Course Analysis

4.1 Basics

Table 4 shows important averages for each run of each course: how many learners register, view, explore, complete, and are repeating the course. The percentages reported are the percent of total learners registered who viewed, explored, and completed the course during that run. The averages for 1x and 2x across all runs are also shown. Although there is some variance in raw numbers across runs, many of the percentages are in the same range. Some of this variance can be explained by edX promo strategies before certain runs of the course (see Section 5.1) and the experiment of an self-paced run as opposed to the usual instructor-led version of the course (see Section 4.2).

In Table 4, we didn’t remove learners who never viewed as we normally do before any analysis. The reason for this is simply to see raw registration numbers to gauge initial popularity of a course in each semester, and how motivated learners who enroll tend to be. After removing learners who never viewed the course for each semester, the average completion rate for a 1x semester was 4.7% and 7.0% for a 2x semester.

There have been 1,190,842 unique people who have signed up for 1x or 2x (or both) over the past 6 years. Around 760,000 of those are considered active learners, meaning 760,000 unique learners have viewed the course. The average learner takes 1.28 semesters worth of courses. The average age of learners is 28.8, and 14% of learners are female. The average grade of learners who pass the course is 84.3%; the average grade of learners who complete at least one assignment (i.e. earn a nonzero overall grade) is 20.9%.

As we saw in Section 1.4, each assignment has a different weight that contributes to the overall grade. The passing grade for each course is only 55% of the total points, so learners

---

5As a reminder, learners who view the course log onto the course at least once after signing up. Learners who explore visit at least half the chapters in the course. Learners who complete earn an overall grade of at least 55%.
<table>
<thead>
<tr>
<th>term</th>
<th>registered</th>
<th>viewed</th>
<th>explored</th>
<th>complete</th>
<th>repeaters</th>
</tr>
</thead>
<tbody>
<tr>
<td>3T2013</td>
<td>79,688</td>
<td>57,299 / 72%</td>
<td>5309 / 6.7%</td>
<td>4974 / 6.2%</td>
<td>0 / 0.0%</td>
</tr>
<tr>
<td>2T2014</td>
<td>33,377</td>
<td>22,332 / 67%</td>
<td>4520 / 14%</td>
<td>2307 / 6.9%</td>
<td>5122 / 15%</td>
</tr>
<tr>
<td>3T2014</td>
<td>31,056</td>
<td>21,083 / 68%</td>
<td>5155 / 17%</td>
<td>2108 / 6.8%</td>
<td>6798 / 22%</td>
</tr>
<tr>
<td>1T2015</td>
<td>76,693</td>
<td>50,459 / 66%</td>
<td>7903 / 10%</td>
<td>4718 / 6.1%</td>
<td>13,368 / 17%</td>
</tr>
<tr>
<td>2T2015</td>
<td>96,361</td>
<td>54,821 / 57%</td>
<td>7367 / 7.6%</td>
<td>5109 / 5.3%</td>
<td>14,489 / 15%</td>
</tr>
<tr>
<td>3T2015</td>
<td>90,074</td>
<td>55,626 / 62%</td>
<td>5525 / 6.1%</td>
<td>3280 / 3.6%</td>
<td>18,070 / 15%</td>
</tr>
<tr>
<td>1T2016</td>
<td>115,559</td>
<td>64,986 / 56%</td>
<td>6507 / 5.6%</td>
<td>3660 / 3.2%</td>
<td>23,451 / 20%</td>
</tr>
<tr>
<td>2T2016</td>
<td>40,632</td>
<td>22,084 / 54%</td>
<td>2695 / 6.6%</td>
<td>1784 / 4.4%</td>
<td>10,060 / 25%</td>
</tr>
<tr>
<td>3T2016</td>
<td>112,884</td>
<td>63,916 / 57%</td>
<td>11,330 / 10%</td>
<td>4223 / 3.7%</td>
<td>23,766 / 21%</td>
</tr>
<tr>
<td>1T2017</td>
<td>69,239</td>
<td>45,094 / 65%</td>
<td>8138 / 12%</td>
<td>3050 / 4.4%</td>
<td>19,580 / 28%</td>
</tr>
<tr>
<td>2T2017</td>
<td>74,985</td>
<td>44,291 / 59%</td>
<td>7645 / 10%</td>
<td>3202 / 4.3%</td>
<td>16,898 / 23%</td>
</tr>
<tr>
<td>3T2017</td>
<td>340,529</td>
<td>225,985 / 66%</td>
<td>20,224 / 5.9%</td>
<td>6744 / 1.9%</td>
<td>38,531 / 11%</td>
</tr>
<tr>
<td>3T2018</td>
<td>51,563</td>
<td>32,842 / 64%</td>
<td>4581 / 8.9%</td>
<td>1574 / 3.1%</td>
<td>12,657 / 25%</td>
</tr>
<tr>
<td>1T2019</td>
<td>57,258</td>
<td>35,736 / 62%</td>
<td>4237 / 7.4%</td>
<td>1246 / 2.2%</td>
<td>12,921 / 23%</td>
</tr>
<tr>
<td>2T2019</td>
<td>54,615</td>
<td>31,283 / 57%</td>
<td>4995 / 9.1%</td>
<td>1704 / 3.1%</td>
<td>13,774 / 25%</td>
</tr>
<tr>
<td>3T2019</td>
<td>51,250</td>
<td>29,131 / 57%</td>
<td>3590 / 7.0%</td>
<td>1108 / 2.2%</td>
<td>14,794 / 29%</td>
</tr>
<tr>
<td><strong>1x total</strong></td>
<td><strong>1,375,753</strong></td>
<td><strong>856,968 / 62%</strong></td>
<td><strong>109,718 / 8.0%</strong></td>
<td><strong>50,792 / 3.7%</strong></td>
<td><strong>244,279 / 17%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>term</th>
<th>registered</th>
<th>viewed</th>
<th>explored</th>
<th>complete</th>
<th>repeaters</th>
</tr>
</thead>
<tbody>
<tr>
<td>3T2014</td>
<td>19,781</td>
<td>13,195 / 67%</td>
<td>2521 / 13%</td>
<td>1501 / 7.6%</td>
<td>0 / 0.0%</td>
</tr>
<tr>
<td>2T2014</td>
<td>20,179</td>
<td>11,772 / 58%</td>
<td>715 / 3.5%</td>
<td>1260 / 6.2%</td>
<td>2949 / 15%</td>
</tr>
<tr>
<td>3T2015</td>
<td>24,982</td>
<td>14,941 / 60%</td>
<td>404 / 1.7%</td>
<td>1586 / 2.3%</td>
<td>4715 / 19%</td>
</tr>
<tr>
<td>1T2015</td>
<td>20,509</td>
<td>12,524 / 61%</td>
<td>289 / 1.4%</td>
<td>1491 / 7.3%</td>
<td>3907 / 19%</td>
</tr>
<tr>
<td>3T2015</td>
<td>22,018</td>
<td>14,287 / 65%</td>
<td>314 / 1.4%</td>
<td>1164 / 5.3%</td>
<td>4262 / 19%</td>
</tr>
<tr>
<td>1T2016</td>
<td>18,346</td>
<td>10,753 / 57%</td>
<td>1890 / 10%</td>
<td>1078 / 5.9%</td>
<td>4473 / 24%</td>
</tr>
<tr>
<td>3T2016</td>
<td>21,412</td>
<td>12,185 / 57%</td>
<td>1575 / 7.4%</td>
<td>739 / 3.5%</td>
<td>4584 / 21%</td>
</tr>
<tr>
<td>3T2017</td>
<td>37,107</td>
<td>20,767 / 56%</td>
<td>2164 / 5.8%</td>
<td>1225 / 3.3%</td>
<td>6151 / 16.6%</td>
</tr>
<tr>
<td>3T2018</td>
<td>7703</td>
<td>4440 / 58%</td>
<td>545 / 7.1%</td>
<td>328 / 4.3%</td>
<td>2289 / 30%</td>
</tr>
<tr>
<td>1T2019</td>
<td>7668</td>
<td>3638 / 47%</td>
<td>572 / 7.5%</td>
<td>316 / 4.1%</td>
<td>1987 / 26%</td>
</tr>
<tr>
<td>3T2019</td>
<td>8726</td>
<td>3924 / 45%</td>
<td>762 / 8.7%</td>
<td>480 / 5.5%</td>
<td>1957 / 22%</td>
</tr>
<tr>
<td><strong>2x total</strong></td>
<td><strong>208,431</strong></td>
<td><strong>122,426 / 59%</strong></td>
<td><strong>11,751 / 5.6%</strong></td>
<td><strong>11,168 / 5.4%</strong></td>
<td><strong>37,274 / 18%</strong></td>
</tr>
</tbody>
</table>

Table 4: The table above reports stats on the number and percentages of learners who register, view, explore, complete, and are repeating each run of the 1x (top) and 2x (bottom) courses. The percentages shown indicate the percent of total learners registered who viewed, explored, completed, and repeated the course.
do not have to complete or even attempt every assignment in order to pass the course. In fact, 15.5% of learners who complete a course will not even attempt the final. The final is only worth 25% of the overall grade, and because many learners have earned over 55% of the points by the time the final comes around, many learners decide not to take it. However, most learners who complete the course at least attempt all the other assignments. Only 2% of learners who complete do not take the midterm, 0.01% of learners who complete the course attempt none of the psets, and 0.1% of learners who complete the course attempt none of the finger exercises.

4.2 Self-paced vs. Instructor-led

edX has two modes to run a course in: self-paced or instructor-led. The instructor-led run is open for about two months while the self-paced run is open for an entire year. During the twelve months of a self-paced run, a learner is expected to complete all the graded problem sets and quizzes on their own time. There are no deadlines in self-paced; whatever they have completed by the course end date is what is used to calculate their final grade. So in theory, a learner could complete all the assignments in the final week of a course and still receive credit. On the other hand, the instructor-led version has staggered release dates for content and deadlines for all assignments besides finger exercises. Learners who complete material after these deadlines don’t receive credit. An example of the schedule for assignments is shown in Table 1 in Section 1.4. Usually, learners have a couple weeks to complete problem sets and until the end of the course to complete finger exercises. Learners have a five day window to complete assessments, but once they start an assessment, they must complete it within 8 hours.

As we saw in Figure 1, most of 1x and 2x is instructor-led. Both 1x and 2x have had one self-paced run, both in the fall of 2017. Because the course changed significantly in the fall of 2016 with the switch from Python 2 to Python 3, we will be comparing the aggregate data from instructor-led runs from 2016 to 2019 with the data from the single self-paced run.
Referring back to Table 4, the number of learners for 1x who registered for the self-paced version of the course was over three times the usual number of learners. Part of this increase in popularity for this particular semester is due to the increased promo emails for 1x by edX (see Section 5.1). Of the more than 300,000 learners who registered for the 1x self-paced course, 66% actually viewed the course, which is on-par with other semesters. Only 5.9% explored the course, which is on the low-end of the range over the semesters but still not the lowest. However, only 1.9% actually completed the course, which is one of the lowest completion rates. This trend of viewed/explored/completed percentages indicates that, without concrete deadlines, students struggle to actually finish the course on-time.

After cleaning the data as described in Section 2.1, we still see a similar trend in completion rates. In 1x, 3.0% of learners in the self-paced and 5.7% of learners in the instructor-led completed the course. In 2x, 5.9% of learners in the self-paced and 8.4% of learners in the instructor-led completed. Table 5 compares the grade, number of active days, number of videos watched, and number of forum events for self-paced and instructor-led learners who completed the course. In 1x, self-paced learners who completed earned an average of 83.4% over 62.6 active days while instructor-led learners who completed earned an average of 85.6% over 51.4 active days. More days active seems like a positive metric on its own, but given that self-paced learners are also scoring lower than their instructor-led counterparts in 1x, it may indicate that self-paced learners are less effective with their time than instructor-led.

<table>
<thead>
<tr>
<th></th>
<th>grade</th>
<th>active days</th>
<th># videos</th>
<th># forum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x Self-paced</td>
<td>83.4%</td>
<td>62.6</td>
<td>67.7</td>
<td>514</td>
</tr>
<tr>
<td>1x Instructor-led</td>
<td>85.6%</td>
<td>51.4</td>
<td>62.1</td>
<td>883</td>
</tr>
<tr>
<td>2x Self-paced</td>
<td>86.4%</td>
<td>51.2</td>
<td>36.6</td>
<td>1221</td>
</tr>
<tr>
<td>2x Instructor-led</td>
<td>86.0%</td>
<td>54.1</td>
<td>33.8</td>
<td>1388</td>
</tr>
</tbody>
</table>

Table 5: The table above compares metrics of learners who complete the self-paced run vs. learners who complete in the instructor-led run. We see different trends for 1x and 2x.
learners: they are spending more time and understanding the material less thoroughly. Interestingly, we see a slightly different trend for 1x than 2x. Instructor-led learners earned a slightly lower average grade over more active days than self-paced learners. This suggests the self-paced mode of the course affects introductory courses more than advanced courses.

Perhaps the most significant analysis is looking at repeat learners of the course who took the course during the self-paced run. This group of repeat learners have taken one or more instructor-led versions of the course before taking the self-paced run (Group A). We compare this group with repeat learners who have taken two or more instructor-led versions of the course and did not take the self-paced run (Group B). Figure 10 explores the completion rates of Groups A and B. We examine the completion rates in the self-paced run, the average completion rate over the instructor-led runs, and the overall completion rates of the groups (i.e. if a student ever completed the course in any run). For both 1x and 2x, overall a lower percent of learners in Group A eventually complete the course than learners in Group B.

![Figure 10: This graph compares completion rates for repeater learners who have taken both self-paced and instructor-led runs (Group A) and repeat learners who only took instructor-led runs (Group B). We examine the percent of learners who eventually complete in any semester (ever), the percent of learners who complete during the self-paced run (self), and the percent of learners who complete during an instructor-led semester (instructor).](image)
The fact that fewer repeaters complete the course in the self-paced run is to be expected, given the overall lower completion rates across all students. However, the fact that repeaters who take the self-paced run are almost half as likely to complete the course at some point is surprising. The reason is actually because learners are much less likely to retake the course after taking the self-paced run. Only 4.7% of learners retook the course after taking the self-paced run while 14.1% on average retook the course after an instructor-led run. A similar trend is found in 2x, with repeaters being 6% less likely to retake the course after taking the self-paced run. After taking a self-paced run, learners realize that if they can’t complete the course over a year, then they likely will never even try again in a subsequent 9-week long run.

One disadvantage of self-paced vs. instructor-led is learners aren’t working on the same problems at the same time. From anecdotal evidence from the CTAs during the self-paced version of the course, the forum was extremely quiet during the self-paced runs. For context, the sum of all the forum posts in the three instructor-led 1x semesters preceding the self-paced course was 32,386 while the self-paced course had a total of only 24,516 posts. We also see this sentiment reflected in Table 5 — forum activity among learners who completed the course is much lower in self-paced runs, especially for 1x.

### 4.3 Intro vs. Advanced

As stated in Section 1.4, 1x is considered an introductory course while 2x is considered an advanced course. This is largely because 1x requires no prior knowledge while 2x does require familiarity with Python and basic computer science concepts, and therefore the material is harder in 2x.

The first clear difference between 1x and 2x is enrollment numbers. Because it’s an introductory course, 1x is always more popular and usually has about 5 times as many
learners as 2x. The average number of active learners each a semester of the course is around 56,700 for 1x and 11,800 for 2x. Because of this, we have a much larger population size for 1x data than 2x data.

The average age of learners across 1x runs was 28.7 with a standard deviation of 10.3. Across 2x runs, the average age was 30.3 with a standard deviation of 10.3. Figure 11 shows a more detailed breakdown of age ranges for each course. We see 1x truly captures the pre-college and college age demographic, while 2x skews more heavily to the older age groups. In every bin beyond age 25, 2x always beats out 1x in overall percent of learners. In 1x, 15% of learners were female and in 2x 11% were female.

One reason instructors started offering 1x during the summer was because they were trying to engage a younger population of junior high schoolers and high schoolers. Of those who provided their age, the spring term of 1x had an average age of 29.3; the summer term had an average of 28.3; and the fall term had an average age of 28.5. The percentage of young people was also calculated for each term. A young person was defined as someone

![Age distribution of learners in 1x and 2x](image)

Figure 11: The age distribution of active learners in 1x and 2x. 1x tends to have younger learners and 2x tends to have older learners.
with a junior high school level of education or a high school level of education between the ages of 12 and 19. Of those who provided their education level, the spring term of 1x was 5.8% young learners, the summer term was 8.1% young learners; and the fall term was 5.3% young learners. These statistics suggest that the summer captures more learners in a younger demographic, but it’s not very statistically significant.

For 1x and 2x, the average grade of those who completed the course was 84.5% and 84.8% respectively; the grades for 1x had a standard deviation of 14.0% and the grades for 2x had a standard deviation of 12.9%. Figure 12 shows the grade breakdown of learners in 1x and 2x. The overall grade and pset averages are almost identical for the two courses. The finger exercises average and final are noticeably higher for 2x; the midterm is noticeably lower for 2x. Usually the final exam averages are skewed by learners who don’t take the final (which

---

7 As a reminder, finger exercises are short exercises between videos, with solutions available in-course.

Figure 12: The average grades learners who complete the course earn in each assignment category in 1x and 2x.
is a non-negligible percentage, as we saw in Section 4.1. If we consider only learners who completed the course and attempted the final, 2x learners score one point worse on average on the final, not 3 points better as we saw in Figure 12. This is due to the fact that 16.5% of 1x learners who pass the course don’t take the final while only 11.6% of 2x learners who pass the course don’t take the final. The higher completion rate of the final and the higher grade on finger exercises (which are usually relatively straightforward and easy to score high on) suggests 2x learners may be more motivated by learning rather than grade than their 1x counterparts.

Throughout the entire paper, many results have been reported with 1x and 2x already separated. Sometimes, 1x and 2x follow similar trends like in Figure 5 in Section 3.1 which looked at learners’ problem set vs. exam grades and Figure 7 also in Section 3.1 which showed completion rates with respect to learner signup dates. Other times, the behavior of learners in 1x and 2x can be very different. Section 5.3 showed almost opposite effects of gating on 1x and 2x, and Section 4.2 showed 2x was less negatively affected by self-paced runs than 1x. In general, 1x and 2x are very similar in course structure, subject material, and even the pool of learners, so it makes sense to see similar trends in some areas. Even so, the nuances of each course, such as 2x’s more advanced nature and 1x’s much higher registration numbers, can lead to differences that shouldn’t be overlooked.

5 Monetary Analysis

As with any business, edX needs to be concerned about how much money is being made. Since it’s establishment in 2012, edX has made a few changes in the hopes of bringing in more money. These include email promos of popular courses (usually sent to learners who have taken other courses on edX), increasing the certification fee, and, most recently, providing only verified users with access to certain features and course content. We will analyze how these changes affect potential profit, the number of verified users, and how learners interact
with the course in 1x and 2x.

5.1 edX Promos

Throughout the year, edX will send emails to learners on the platform about popular courses that will be offered in the coming weeks or months. In general, a course is usually advertised about 3 times before or right after the start date. Graph [13] shows the few semesters that 1x was promoted as well as the registration numbers. The 3T2017 semester, which was the self-paced run of the course, was highly advertised which contributed to its peak enrollment numbers. The self-paced run was also open for an entire year, which is part of the reason why it was more advertised. Taking the highly advertised, highly enrolled self-paced semester from consideration, there seems to be no clear correlation between number of promo emails and number of learners who register for the course.

**Effect of promo emails on registration numbers**

![Figure 13: This graph shows the relationship between number of times a semester of 1x is promoted and the number of learners registered for that semester. For 1x, a semester is promoted anywhere between 1 and 19 times.](image)
Data about how often promo emails were sent out is available from 2016 on, so unfortunately there isn’t a large sample size for analysis, especially for 2x. However, it could be very helpful to analyze this data going forward to see how effective promo emails really are in encouraging learners to take courses. We also have no data about courses being advertised outside the promo emails sent by edX; external factors probably also affect enrollment numbers semester to semester. For example, 1x is featured in the top 100 highest-rated MOOCs by Class Central [7]. Promotions like this outside edX can help to drive traffic to the course and increase registration numbers.

5.2 Increase in Certification Fee

In 2018 between the summer and fall semesters, edX increased the certification cost from $50 to $75 per course. It’s important to note time- and feature-gating that was introduced in 2019 will also significantly affect certification numbers (see Section 5.3). Because of this, there is only one semester – the fall 2018 semester of 2x – where we can analyze the effect of the cost increase before gating was implemented.

In all the semesters prior to 2018 in 2x, 5.3% of learners purchased certification. In fall of 2018, the semester after the fee was increased by $15, 4.7% of learners opted to become verified users. Given this is only a 12% decrease of users purchasing certification, while the fee increase was 50%, edX is likely happy with these numbers as they are making more money even though fewer learners are opting to become verified. Verified learners who paid the higher fee also might be more motivated to finish – the award rate for certified users before the increase was 69.5% while the award rate after the increase was 76.4%.

Overall, there seems to be a dip in the number of verified users and an increase in award rate with the increase in certification fee, but this analysis is only based on one semester of 2x because of the gating changes made in 2019.
5.3 Time- and Feature-Gating

Time and feature gating were introduced by edX beginning in the summer terms of 2019 in an effort to generate more revenue from many of their MOOCs. Time-gating is restricting access to the course after the course ends. Only verified learners who pay the certification fee can review the course material more than a few months after the course ends. Feature-gating is restricting certain features or content during the course run from learners who are not certified. For example, in 1x and 2x, only verified learners can view and complete the midterm and final exams. Feature-gating in particular will impact the way many learners interact with the course and whether or not learners opt to pay for certification status. As with Section 4.2, we will only be considering non-gated courses past the fall of 2016 when the course was revamped. In the following, we compare the semesters from fall of 2016 to fall of 2018 with the semesters from spring 2019 on.

Perhaps the most drastic change is that unverified learners can only complete pset and finger exercises. As the two exams make up 50% of a learner’s final grade, this means that unverified learners can no longer earn a passing grade in the course. Figure 14 shows the problem set completion and finger exercises completion of non-verified learners before and after gating in 1x. We analyze both the completion rates of these assignments among all learners and the assignment completion rates of learners who completed the first pset and first finger exercises. Across the board, completion rates decreased after gating. But even among learners who attempted the first assignments, the completion rates of subsequent assignments dropped off at a quicker rate after gating than before gating. Clearly, feature-gating hinders non-verified learners motivation to complete the parts of the course they still have access to. The dropoff for finger exercises was less severe than that of problem sets, likely because finger exercises are shorter in length and less effort over all. One way to keep non-verified learners still engaged in gating courses could by adding more short assignments. It’s more likely that non-verified learners will complete these shorter assignments than longer psets if they know the assignments will not contribute to their passing of the course either
Figure 15 shows the verified rates and completion rates for learners in 1x and 2x. The verified rates went up for both 1x and 2x, which was likely edX’s goal. The award rate (the percent of verified learners who pass the course) and the overall completion rate both decreased for 1x, but both increased for 2x.

Considering the above, feature-gating may actually be a positive change in 2x. Both the overall completion rate and the award rate increased after the introduction of feature-gating. This is likely because gating the course encouraged more learners to sign up for verification. As we saw in Section 3.5, verified learners are more motivated in general than unverified learners. However, after gating, we didn’t see that trend in 1x. The verified rate went up,

**Assignment completion of non-verified learners before after gating**

![Graphs showing assignment completion rates before and after feature-gating](image)

Figure 14: The above graphs plot the completion rates of problem sets (left) and finger exercises (right) of unverified learners before and after gating was introduced in 1x. The top graphs show overall completion rates and the bottom graphs show completion rates of learners who attempted the first pset and first finger exercises.
Figure 15: This figure shows the verified and completion rates before and after feature-gating was introduced in 1x (top) and 2x (bottom). We observe opposite trends in the two courses – completion and award rates decrease in 1x and increase in 2x.

but the award rate went down, suggesting learners who weren’t prepared for the course might have been forced into verification status when they weren’t fully committed. This suggests that feature-gating may be a good choice for more advanced courses, where learners have likely taken other MOOCs before and have an idea of what’s expected, but could hinder first-time learners in introductory courses.

6 Conclusions

6.1 Further Work

There are many opportunities for further work related to this research. The easiest is probably applying this analysis to other edX MOOCs, as edX collects the same statistics for all courses. The data visualization dashboard especially can easily be used by be other courses and updated as more and more runs are offered. We have been working with the edX biology instructors throughout this process, and they are excited about potentially using the dashboard for their own data. Hopefully with their support, it will also spread to other courses within edX.

There are also many potentially interesting topics to dive deeper into within 1x and
2x. An entire paper could be written on the forum activity alone – using natural language processing to determine what posts are actually saying, determining how different or similar posts are from semester to semester, and how forum activity and content affect learner performance. As edX focuses more on monetary success, diving further into the effects of gating and course pacing (self-paced vs. instructor-led) on learner performance could help edX understand what works and what doesn’t in these new types of courses.

6.2 Final Thoughts

As we saw throughout this paper, a myriad of factors affect learner performance and the success of a course. Verified and repeat learners have the best chance of passing, but simply completing problem sets, signing up early, and being more active in a course will set any learner up for success.

While edX continues to time- and feature-gate more courses, we give the recommendation to focus on gating advanced courses while keeping introductory courses open to all, as 2x actually responded positively to gating. We offer a similar recommendation for self-paced vs. instructor-led courses – 1x is affected much more negatively by changing to self-paced runs than 2x is.

Hopefully, this paper and the code attached can provide a starting point for any other analyst trying to understand learner engagement in their MOOCs. From parsing and cleaning the data to analyzing and visualizing that data in a meaningful and unbiased way, being intentional and careful at every step is critical. Better understanding of every aspect of these MOOCs and their learners is the first step in providing the best online education experience for all.
7 References

www.news.mit.edu/2012/mit-harvard-edx-announcement-050212


2019.

MOOC. May 2019.

”Multiple-Account” Cheating in Massive Open Online Courses. August 2018.

https://www.doorkeeper.jp/event-planning/increasing-participants-decreasing-no-shows

[7] Top 100 Free Online Courses. Class Central, 
https://www.classcentral.com/collection/top-free-online-courses?utm_ 
source=top-50-course-page

[8] Mitodl/edx2bigquery. GitHub, 
www.github.com/mitodl/edx2bigquery/tree/master/edx2bigquery/schemas
Glossary

1x  edX’s Introduction to Computer Science and Programming Using Python online course.

2x  edX’s Introduction to Computational Thinking and Data Science online course.

active learner  A learner who at least logs onto a course once.

certified learner  A verified learner who completes the course with a passing grade and is able to download a certificate of completion.

completing a course  When a learner earns a passing grade of 55% or above in a course.

CTA  aka Communitiy Teaching Assistant. Volunteers who help moderate the forum.

dual-course learner  A learner who takes both 1x and 2x at least once.

exploring a course  When a learner visits at least half of the chapters in a course.

feature-gating  Restricts non-verified learners’ access to course content, such as the midterm and final exams.

finger exercise  A very short assignment associated with each lecture video.

forum  The online platform within the course where students can ask and answer questions, read others’ posts, and interact with other students and CTAs.

instructor-led  A course that has specific deadlines and release dates for assignments; learners must complete assignments by due dates or they don’t receive credit.

MOOC  aka Massive Open Online Course, where virtually anyone can enroll and learn about a topic.
**pset** aka problem set. A multiple week assignment where learners tackle a large coding problem.

**repeat cross-referencer** A learner who accesses at least two previous courses in a later course. These learners are potentially using course and forum content from previous semesters to complete current semesters.

**repeat learner** A learner who has taken the same course more than once.

**self-paced** A course that has no deadlines or release dates for assignments; learners simply start and complete assignments at their discretion.

**time-gating** Restricts non-verified learners’ access to a course a few months after the course ends.

**verified learner** A learner who pays a $75 fee to earn a certification if they complete the course. In the most recent semesters, being a verified learner also grants access to certain course content.

**viewing a course** When a learner logs onto the course at least once. See active learner.
Appendices

A Python Scripts

The code is split up into multiple files in an effort to keep things organized. The central file is main.py, which loads and cleans the data and calls the main function of the other scripts. The main.py script uses importlib to reload other scripts without terminating the session. This helped with efficiency because we could perform different analysis and execute the new code without having to re-parse and re-clean the huge datasets again.

All the code below and more is available on GitHub at https://github.com/vonderhaar/MEng-thesis. The files here include the main pre-processing of the data, the code for the different classes described in section 2.3 and one analysis script. More analysis not included here is in the GitHub repository. Because of user confidentiality, the data cannot be made available to the public.

main.py

```python
import sys
from load_data import load_course
from classes import Course, Cohort, Semester, Student
from collections import defaultdict
import clean_data, classes, load_data, monetary_analysis, repeaters,
    general, course_analysis, learner_analysis
import importlib, traceback

course_names_1x = ['1 x_3T2013', '1 x_2T2014', '1 x_3T2014', '1 x_1T2015', '1 x_2T2015', '1 x_3T2015', '1 x_1T2016', '1 x_2T2016', '1 x_3T2016', '1 x_1T2017', '1 x_2T2017', '1 x_3T2017', '1 x_3T2018', '1 x_1T2019', '1 x_2T2019', '1 x_3T2019']
course_names_2x = ['2 x_1T2014', '2 x_3T2014', '2 x_1T2015', '2 x_3T2015', '2 x_1T2016', '2 x_3T2016', '2 x_1T2017', '2 x_3T2017', '2 x_3T2018', '2 x_1T2019', '2 x_3T2019']
self_paced_1x = '1 x_3T2017'
self_paced_2x = '2 x_3T2017'
```
# set promo data
promo_1x = defaultdict(int)
promo_1x['1 x_3T2016'] = 3
promo_1x['1 x_1T2017'] = 3
promo_1x['1 x_2T2017'] = 3
promo_1x['1 x_3T2017'] = 19
promo_1x['1 x_3T2018'] = 2
promo_1x['1 x_1T2019'] = 1
promo_1x['1 x_2T2019'] = 3
promo_1x['1 x_3T2019'] = 4
promo_2x = defaultdict(int)
promo_2x['2 x_1T2017'] = 1
promo_2x['2 x_3T2017'] = 2
promo_2x['2 x_3T2018'] = 1

# create courses objects
course_1x_raw = Course(course_names_1x, self_paced_1x, True, promo_1x)
course_2x_raw = Course(course_names_2x, self_paced_2x, False, promo_2x)
reclean = True
loaded = False

while True:
    try:
        # Load and clean data
        if not loaded:
            load_course(course_1x_raw)
            load_course(course_2x_raw)
            loaded = True
        if reclean:
            course_1x, ctas_1x, cheaters_1x = clean_data.clean(course_1x_raw)
            course_1x.set_ctas(ctas_1x)
            course_1x.set_cheaters(cheaters_1x)
            course_2x, ctas_2x, cheaters_2x = clean_data.clean(course_2x_raw)
            course_2x.set_ctas(ctas_2x)
            course_2x.set_cheaters(cheaters_2x)

        # Perform analysis
        general.main(course_1x, course_2x)
        repeaters.main(course_1x, course_2x)
        course_analysis.main(course_1x, course_2x)
        monetary.main(course_1x, course_2x)
        learner_analysis.main(course_1x, course_2x)

    except Exception:
        traceback.print_exc()

    response = input("\n\nEnter R to use reclean, any other key to rerun, CTRL-C to exit: ")
    reclean = True if response.upper() == 'R' else False

    # Reload scripts and potentially reclean data
    importlib.reload(classes)
    importlib.reload(general)
    importlib.reload(repeaters)
importlib.reload(course_analysis)
importlib.reload(learner_analysis)
importlib.reload(monetary)
if reclean:
    importlib.reload(clean_data)

load_data.py

import csv
import copy
import sys
from collections import defaultdict
import datetime
from classes import Course, Cohort, Semester, Student, Entry
import traceback

username_userid_dic = defaultdict(str)

# Make Cohort mapping Student -> Entry

def make_repeaters(course):
    repeaters = Cohort()
    for entry in course.get_all_entries():
        repeaters.add_entry(entry)
    return repeaters


def load_course(course):
    dates_map = make_course_run_dates(course)
    counter = 0
    for sem_name in course.semester_names:
        semester = Semester(sem_name, course.is_1x, sem_name == course.
        self_paced_run, dates_map[sem_name])

        # load person course data
        with open('/Volumes/Research/person_course_data/6.00.' + sem_name + '._person_course.csv') as csv_file:
            data = csv.reader(csv_file, delimiter=',')
            for row in data:
                if row[0] != 'course_id':  # skip header row
                    entry = make_entry(row, sem_name, dates_map[sem_name][0])
                    semester.add(entry.userid, entry)

        # load grades data
        # We put in a try/except block because we don't have grades data for
        # two semesters of 1x
        try:
            with open('/Volumes/Research/grade_report_data/6.00.' + sem_name + '._grade_report.csv') as csv_file:
                data = csv.reader(csv_file, delimiter=',')
                keys = None
                for row in data:
                    if row[1] == 'Email':  # skip header row
                        index_of_first = -1

49
index_of_final = -1
for i, cell in enumerate(row):
    if 'Grade' in cell:
        index_of_first = i
    if 'Final' in cell:
        index_of_final = i
keys = row[index_of_first : index_of_final +1]
else:
    grades = {}
    for i in range(len(keys)):
        grades[keys[i]] = None if row[i+3] == "Not Attempted" else float(row[i+3])
    entry = semester.get(row[0])
    if entry != None:
        entry.set_grades(grades)
except Exception:
    print(sem_name)
    traceback.print_exc()

# load forum data
with open('/Volumes/Research/forum_data/6.00.' + sem_name + '_forum_events.csv') as csv_file:
    data = csv.reader(csv_file, delimiter=',')
    for row in data:
        if row[0] != 'username':
            d = defaultdict(int)
            d['read'] = 0 if (row[1] == 'null' or row[1] == '') else int(row[1])
            d['original_posted'] = 0 if (row[1] == 'null' or row[1] == '') else int(row[2])
            d['followed'] = 0 if (row[1] == 'null' or row[1] == '') else int(row[3])
            d['wrote'] = 0 if (row[1] == 'null' or row[1] == '') else int(row[4])
            total = d['read'] + d['followed'] + d['wrote']
            entry = semester.get(username_userid_dic[row[0]])
            if entry != None:
                total = d['read'] + d['followed'] + d['wrote']
                entry.set_num_forum_events(total, d)

    # add the Semester to the Course
    course.add(sem_name, semester)

# Make entry object from user data in person_course
def make_entry(row, sem_name, start_date):
    userid = row[1]
    username = row[2]
    viewed = row[4] == 'true'
    explored = row[5] == 'true'
    certified = row[6] == 'true'
    completed = row[7] == 'true'
    education_level = None if row[22] == '' else row[22]
    age = None if row[23] == '' else (start_date - datetime.datetime(int(row[23]), 7, 2)).days/365 # July 2nd is the middlemost day of the year
grade = 0 if row[25] == '' else float(row[25])
start_time = None if row[27] == '' else datetime.datetime.strptime(row[27][:19], '%Y-%m-%d %H:%M:%S')
first_event = None if row[28] == '' else datetime.datetime.strptime(row[28][:19], '%Y-%m-%d %H:%M:%S')
last_event = None if row[29] == '' else datetime.datetime.strptime(row[29][:19], '%Y-%m-%d %H:%M:%S')
um_events = 0 if row[30] == '' else int(row[30])
um_active_days = 0 if row[31] == '' else int(row[31])
um_videos = 0 if row[61] == '' else int(row[61])
um_problem_checks = 0 if row[41] == '' else int(row[41])
mode = None if row[44] == '' else row[44]
username_userid_dic[username] = userid
classes.py

return Entry(sem_name, userid, username, viewed, explored, certified, completed,
             education_level, age, gender, grade, start_time, first_event,
             last_event,
             num_events, num_active_days, num_videos, num_problem_checks, mode)

# Read course start and end dates from .txt files and determine which season (spring, summer, or fall)
def make_course_run_dates(course):
dates_map = {}
file = 'runs_6001x.txt' if course.is_1x else 'runs_6002x.txt'
index = 0
is1x = file == 'runs_6001x.txt'

print(course.semester_names)
with open('/Volumes/Research/admin_data/'+file) as f:
    lines = f.readlines()
    for i in range(0, len(lines), 4):
        start_date = datetime.datetime.strptime(lines[i+1][: -5], 'Course Start Date: %b %d, %Y %H:%M')
        end_date = datetime.datetime.strptime(lines[i+2][: -5], 'Course End Date: %b %d, %Y %H:%M')
        # get season
        semester = course.semester_names[index]
        season = "spring"
        if '3T' in semester or ('2T' in semester and (semester[-2:] == "_2" or semester[-1] == "a")):  
            season = "fall"
        elif '2T' in semester:
            season = "summer"
        dates_map[semester] = [start_date, end_date, season]
        index += 1
return dates_map
from collections import defaultdict
import statistics
import copy

class BigGroup(object):
    def __init__(self):
        self.dic = {}

    def add(self, key, value):
        self.dic[key] = value

    def delete(self, key):
        del self.dic[key]

    def get(self, key):
        if key not in self.dic.keys():
            return None
        return self.dic[key]

    def get_keys(self):
        return self.dic.keys()

    def get_all_entries(self):
        ls = []
        for item in self.dic.values():
            ls.extend(item.get_all_entries())
        return ls

    def get_all_userids(self):
        userids = set()
        for item in self.dic.values():
            userids.update(item.get_keys())
        return userids

    def get_average(self, stat, completed_status=None, verified_status=None, overall=True):
        ls = []

        if overall:
            for entry in self.get_all_entries():
                value = entry.get_stat(stat, completed_status, verified_status)
                if value != None:
                    ls.append(value)
            else:
                for item in self.dic.values():
                    ls.append(item.get_all_entries())
value = item.get_average(stat, completed_status, verified_status)
if value != None:
    ls.append(value)
if len(ls) == 0:
    return None
return statistics.mean(ls)

# a collection of Semester objects
class Course(BigGroup):
    def __init__(self, semester_names = None, self_paced_run = None, is_1x = True, promo_dict = {}):
        super().__init__()
        self.semester_names = semester_names
        self.self_paced_run = self_paced_run
        self.is_1x = is_1x
        self.promo_dict = promo_dict
        self.ext = '1x' if self.is_1x else '2x'
        self.ctas = None
        self.cheaters = None

    def __str__(self):
        return "Course 1x" if self.is_1x else "Course 2x"

    def set_ctas(self, ctas):
        self.ctas = ctas

    def set_cheaters(self, cheaters):
        self.cheaters = cheaters

    def delete(self, key):
        del self.dic[key]
        self.semester_names.remove(key)

    def get_semester_by_index(self, index):
        return self.dic[self.semester_names[index]]

    def get_index_by_semester(self, semester_name):
        return self.semester_names.index(semester_name)

    def is_first_or_last_semester(self, semester_name):
        return self.get_index_by_semester(semester_name) == 0 or self.get_index_by_semester(semester_name) == len(self.semester_names) - 1

    def get_num_registered(self):
        return sum([sem.get_num_registered() for sem in self.dic.values()])

    def deep_copy(self):
        new_course = Course(self.semester_names[:], self.self_paced_run, self.is_1x, self.promo_dict)
        for sem in self.dic.values():
            new_course.add(sem.name, sem.deep_copy())
        return new_course
# a collection of Student objects

class Cohort(BigGroup):
    def __init__(self):
        super().__init__()

    def add_entry(self, entry):
        if entry.userid not in self.dic.keys():
            stud = Student(entry.userid)
            stud.add(entry.sem_name, entry)
            self.dic[entry.userid] = stud
        else:
            self.dic[entry.userid].add(entry.sem_name, entry)

    def get_num_semesters(self, userid):
        return len(self.dic[userid].dic)

    # Returns percent of learners who complete/explore/view in ANY semester
    def get_ever(self, stat):
        return sum([item.get_ever(stat) for item in self.dic.values()]) / len(self.dic)

    def deep_copy(self):
        new_cohort = Cohort()
        for student in self.dic.values():
            new_cohort.add(student.userid, student.deep_copy())
        return new_cohort

class SmallGroup(object):
    def __init__(self):
        self.dic = {}

    def add(self, key, value):
        self.dic[key] = value

    def delete(self, value):
        del self.dic[value]

    def get(self, key):
        if key not in self.dic.keys():
            return None
        return self.dic[key]

    def get_keys(self):
        return self.dic.keys()

    def get_num_registered(self):
        return len(self.dic)

    def get_average(self, stat, completed_status=None, verified_status=None):
        ls = []
        for entry in self.dic.values():
            value = entry.get_stat(stat, completed_status, verified_status)
if value != None:
    ls.append(value)
if len(ls) == 0:
    return None
return statistics.mean(ls)

def get_all_entries(self):
    ls = []
    for entity in self.dic.values():
        ls.append(entity)
    return ls

# a collection of Entry objects organized by semester
class Semester(SmallGroup):
    def __init__(self, name, is_1x, is_self_paced, dates_list):
        super().__init__()
        self.name = name
        self.is_1x = is_1x
        self.is_self_paced = is_self_paced
        self.start_date = dates_list[0]
        self.end_date = dates_list[1]
        self.season = dates_list[2]
        self.students = {}

def __str__(self):
    return str(self.students)

def get_num_registered(self):
    return len(self.dic.values())

def deep_copy(self):
    new_sem = Semester(self.name, self.is_1x, self.is_self_paced, [self.
start_date, self.end_date, self.season])
    for entry in self.dic.values():
        new_sem.add(entry.userid, entry)
    return new_sem

# a collection of Entry objects organized by student
class Student(SmallGroup):
    def __init__(self, userid):
        super().__init__()
        self.userid = userid

def get_num_semesters(self):
    return len(self.dic)

def is_in_semester(self, semester_name):
    return semester_name in self.dic.keys()

def get_first_semester(self, semester_names):
first = None
first_index = None
for sem_name in self.dic.keys():
    if first_index == None or first_index > semester_names.index(sem_name):
        first = sem_name
        first_index = semester_names.index(sem_name)
return first

def get_last_semester(self, semester_names):
    last = None
    last_index = None
    for sem_name in self.dic.keys():
        if last_index == None or last_index < semester_names.index(sem_name):
            last = sem_name
            last_index = semester_names.index(sem_name)
    return last

# Returns true/false depending on if the learner completes/explores/views in ANY semester
def get_ever(self, stat):
    for entry in self.dic.values():
        if stat == 'completed' and entry.is_completed():
            return 1
        elif stat == 'explored' and entry.explored:
            return 1
        elif stat == 'viewed' and entry.viewed:
            return 1
    return 0

def deep_copy(self):
    new_student = Student(self.userid)
    for entry in self.dic.values():
        new_student.add(entry.sem_name, entry)
    return new_student

# Represents one learner in one semester
class Entry(object):
    def __init__(self, sem_name, userid, username, viewed, explored,
                 certified, completed,
                 education_level, age, gender, grade, start_time, first_event,
                 last_event,
                 num_events, num_active_days, num_videos, num_problem Checks,
                 mode):
        self.sem_name = sem_name
        self.userid = userid
        self.username = username
        self.viewed = viewed
        self.explored = explored
        self.certified = certified
        self.completed = completed
self.education_level = education_level
self.age = age
self.gender = gender
self.grade = grade
self.start_time = start_time
self.first_event = first_event
self.last_event = last_event
self.num_events = num_events
self.num_active_days = num_active_days
self.num_videos = num_videos
self.num_problem_checks = num_problem_checks
self.mode = mode

# these are set later when different files are read
self.grades = {}
self.forum_dic = defaultdict(int)
self.num_forum_events = None
self.is_cta = False

def get_stat(self, stat, completed_status = None, verified_status = None):
    value = None

    if stat == 'overall_grade':
        value = self.get_grade()
    elif stat == 'pset':
        value = self.get_pset_average()
    elif stat == 'quiz':
        value = self.get_quiz()
    elif stat == 'finger_exercises':
        value = self.get_finger_ex_average()
    elif stat == 'active_days':
        value = self.num_active_days
    elif stat == 'forum':
        value = self.num_forum_events
    elif stat == 'videos':
        value = self.num_videos
    elif stat == 'problem_checks':
        value = self.num_problem_checks
    elif stat == 'age':
        value = self.age
    elif stat == 'female':
        value = 1 if self.gender == 'f' else 0
    elif stat == 'verified':
        value = 1 if self.is_verified() else 0
    elif stat == 'unverified':
        value = 0 if self.is_verified() else 0
    elif stat == 'completed':
        value = 1 if self.is_completed() else 0
    elif stat == 'explored':
        value = 1 if self.explored else 0
    elif stat == 'viewed':
        value = 1 if self.viewed else 0
value = 1 if self.viewed else 0

# if verified_status = None, we don't have to do anything
if (verified_status == 'verified' and not self.is_verified()) or (verified_status == 'non_verified' and self.is_verified()):
    value = None

# if completed_status = None, we don't have to do anything
if (completed_status == 'completed' and not self.is_completed()) or (completed_status == 'not_completed' and self.is_completed()):
    value = None

return value

def is_verified(self):
    return self.mode == 'verified' or self.mode == 'credit'

def is_completed(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015, so we must
    # determine based on the "completed" field. This is valid because the "completed"
    # field is accurate before 2016
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return self.completed
    return self.get_grade() >= 0.55

def set_grades(self, grades_dic):
    self.grades = grades_dic

def set_num_forum_events(self, num, dic):
    self.num_forum_events = num
    self.forum_dic = dic

def get_grade(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    if "Grade" not in self.grades:
        return 0

    return self.grades["Grade"]

def get_number_pset_attempted(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    counter = 0
    for pset in self.get_pset_grades():
        counter += 1 if pset != None else 0
    return counter

def get_number_exercises_attempted(self):
# There is no grade_report information on 1x_3T2013 or 1x_3T2015
if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
    return None

counter = 0
for ex in self.get_finger_ex_grades():
    counter += 1 if ex != None else 0
return counter

def get_finger_ex_grades(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    ls = []
    for i in range(1,13):
        ex = None

        for key in self.grades:
            if "Finger Exercises " + str(i) in key:
                ex = self.grades[key]
                break

        ls.append(ex)
    return ls

def get_finger_ex_average(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    val = 0
    if "Finger Exercises (Avg)" in self.grades:
        val = self.grades["Finger Exercises (Avg)"]
    if "Lecture Sequence (Avg)" in self.grades:
        val = self.grades["Lecture Sequence (Avg)"]
    return val if val != None else 0

def get_pset_grades(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    ls = []
    for i in range(1,7):
        pset = self.grades['Problem Set ' + str(i) + ': Problem Set ' + str(i)]

        # print(pset)
        ls.append(pset)
    return ls

def get_pset_average(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None
val = 0
if "Problem Set (Avg)" in self.grades:
    val = self.grades["Problem Set (Avg)"]
return val if val != None else 0

def get_quiz(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    val = 0
    if "Quiz" in self.grades:
        val = self.grades["Quiz"]
    if 'Midterm' in self.grades:
        val = self.grades["Midterm"]
    return val if val != None else 0

def get_final(self):
    # There is no grade_report information on 1x_3T2013 or 1x_3T2015
    if self.sem_name == '1x_3T2013' or self.sem_name == '1x_3T2015':
        return None

    val = 0
    if "Final" in self.grades:
        val = self.grades["Final"]
    if 'Final Exam' in self.grades:
        val = self.grades["Final Exam"]
    return val if val != None else 0

def __str__(self):
    return self.username

 monetary_analysis.py

 from classes import Course, Cohort, Semester, Student
 from collections import defaultdict
 import statistics
 import csv
 import matplotlib.pyplot as plt

 # splits the course into smaller groups of Semesters
 def split_course(course, start_index, split_index, end_index):
     before = Course(course.semester_names[start_index:split_index], None, course.is_1x)
     after = Course(course.semester_names[split_index:end_index+1], None, course.is_1x)
     for i in range(start_index, end_index+1):
         name = course.semester_names[i]
         if i < split_index:
             before.add(name, course.get(name))
         else:
after.add(name, course.get(name))

return (before, after)

# looks at registration numbers and number of promo emails for each semester
def promo(course):
    filename = './visualization/data/' + course.ext + '/promo.csv'
    with open(filename, mode='w') as file:
        writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        writer.writerow(['Semester', 'Registered', 'TotalPromo'])

        for semester in course.semester_names:
            if course.promo_dict[semester] == 0:
                continue
            registered = course.get(semester).get_num_registered()
            writer.writerow([semester, registered, course.promo_dict[semester]])

    # looks at registration and verified numbers before and after the increase in certification fee
def increase_cert(course, start_index, split_index, end_index):
    before, after = split_course(course, 0, start_index, split_index, end_index)

    print('right before increase', course.get_semester_by_index(split_index -1).get_average('verified'))
    print('right after increase', course.get_semester_by_index(split_index).get_average('verified'))

    print('before increase', before.get_average('verified'))
    print('after increase', after.get_average('verified'))
    print('
before increase', before.get_award_rate())
    print('after increase', after.get_award_rate())

    # looks at learner engagement before and after gating was introduced
def gating(course, start_index, split_index, end_index):
    before, after = split_course(course, start_index, split_index, end_index)

    print(after.semester_names)
    print(before.semester_names)

    print('before gating:')
    print('verified rate', before.get_average('verified', overall = False))
    print('completion rate (verified)', before.get_average('completed',
        overall = False, verified_status = 'verified'))
    print('completion rate (non_verified)', before.get_average('completed',
        overall = False, verified_status = 'non_verified'))
    print('completion rate (overall)', before.get_average('completed',
        overall = False))
    print('pset average', before.get_average('pset', verified_status = 'non_verified'))
print('after gating:')
print('verified rate', after.get_average('verified', overall = False))
print('completion rate (verified)', after.get_average('completed', overall = False, verified_status = 'verified'))
print('completion rate (non_verified)', after.get_average('completed', overall = False, verified_status = 'non_verified'))
print('completion rate (overall)', after.get_average('completed', overall = False))
print('pset average', after.get_average('pset', verified_status = 'non_verified'))

if course.is_1x:
    psets_before = assignment_helper(before)
psets_after = assignment_helper(after)

    finger_before = assignment_helper(before, is_pset = False)
finger_after = assignment_helper(after, is_pset = False)

filename = './visualization/data/' + course.ext + '/gating_unverified.csv'
with open(filename, mode='w') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    writer.writerow(['Before', 'After'])
    for grade_before, grade_after in zip(psets_before[1], psets_after[1]):
        writer.writerow([statistics.mean(grade_before), statistics.mean(grade_after)])

filename = './visualization/data/' + course.ext + '/gating.csv'
with open(filename, mode='w') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    writer.writerow(['Before', 'After'])
    writer.writerow([before.get_average('verified', overall = False), after.get_average('verified', overall = False)])
    writer.writerow([before.get_average('completed', overall = False), after.get_average('completed', overall = False)])
    writer.writerow([before.get_average('completed', overall = False, verified_status = 'verified')/10])
    writer.writerow([before.get_average('completed', overall = False, verified_status = 'non_verified'), after.get_average('completed', overall = False, verified_status = 'non_verified')])

# gets pset/finger excercise completion rates per assignement
def assignment_helper(course, is_pset = True):
    overall = []
    active_learners = []
    for i in range(6 if is_pset else 12):
        overall.append([])
        active_learners.append([])
for entry in course.get_all_entries():
    if entry.grades == {}:
        continue
    if not entry.is_verified():
        first = True
    grades = entry.get_pset_grades() if is_pset else entry.get_finger_ex_grades()
    for i in range(len(grades)):
        grade = grades[i]
        if i == 0 and grade == None:
            first = False
            overall[i].append(1 if grade != None else 0)
        if first:
            active_learners[i].append(1 if grade != None else 0)

return (overall, active_learners)

def main(course_1x, course_2x):
    print("\n------------------------ MONETARY ----------------------\n")
    print('1x results:\n')
    promo(course_1x)
    print('n2x results:\n')
    promo(course_2x)
    # increase_cert(course_2x, 8, 9)
    # gating(course_2x, 5, 10, len(course_2x.semester_names)-1)