Towards DIVE: A platform for automatic visualization and analysis of structured datasets.

by

Kevin Zeng Hu

S.B. Physics, Massachusetts Institute of Technology (2013)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2015

Signature redacted

Author ....
Program in Media Arts and Sciences
August 7, 2015

Signature redacted

Certified by ....
César A. Hidalgo
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Signature redacted

Accepted by ....
Pattie Maes
Academic Head, Program in Media Arts and Sciences

Towards DIVE: A platform for automatic visualization and analysis of structured datasets.

by

Kevin Zeng Hu

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on August 7, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Our world is filled with data describing complex systems like international trade, personal mobility, particle interactions, and genomes. To make sense of these systems, analysts often use visualization and statistical analysis to identify trends, patterns, or anomalies in their data. However, currently available software tools for visualizing and analyzing data have major limitations because they have prohibitively steep learning curves, often need domain-specific customization, and require users to know *a priori* what they want to see before they see it. Here, I present a new platform for exploratory data visualization and analysis that automatically presents users with inferred visualizations and analyses. By turning data visualization and analysis into an act of curation and selection, this platform aspires to democratize the techniques needed to understand and communicate data.

Conceptually, for any dataset, there are a finite number of combinations of its elements. Therefore there are a finite number of common visualizations or analyses of a dataset. In other words, it should be possible to enumerate the whole space of possible visualizations and analyses for a given dataset. Analysts can then explore this space and select interesting results. To capture this intuition, we developed a conceptual framework inspired by set theory and relational algebra, and drawing upon existing work in visualization and database architecture. With these analytical abstractions, we rigorously characterize datasets, infer data models, enumerate visualizable or analyzable data structures, and score these data structures. We implement this framework in the Data Integration and Visualization Engine (DIVE), a web-based platform for anyone to efficiently analyze or visualize arbitrary structured datasets, and then to export and share their results. DIVE has been under development since March 2014 and will continue being development in order to have a alpha version available by September 2015 and a beta version by the end of 2015.

Thesis Supervisor: César A. Hidalgo
Title: Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Towards DIVE: A platform for automatic visualization and analysis of structured datasets.

by

Kevin Zeng Hu

The following people served as readers for this thesis:

Signature redacted

Thesis Reader . . . .

César Hidalgo
Associate Professor of Media Arts and Sciences
MIT Media Lab

Signature redacted

Thesis Reader . . . .

Deb Roy
Associate Professor Media Arts and Sciences
MIT Media Lab

Signature redacted

Thesis Reader . . . .

Ethan Zuckerman
Principal Research Scientist
MIT Media Lab

Acknowledgments

Im thankful to have many people and circumstances to include in these acknowledgements. It's a privilege to work with such wonderful people on interesting problems. My first thanks goes to my family: my mom, dad, and Siss. I'm extremely lucky to have a family that is not only supportive, but also inspiring.

My second thanks goes to my second family, my friends from all places and times. You know who you are and I love you all very much – the CK CompSci lab crew, Suite C, Funfetti. This list includes but is not limited to: Sophie, Charlie, Mike, Franck, Will, Taylor, Andrei, Travis, Jeremy, Thariq, Alexis, Julie, Savannah, Matt.

Also I thank my research group, Macro Connections, and especially my advisor Cesar Hidalgo. Good teachers teach things, great teachers teach how to learn and how to think. In that way, he has been the greatest teacher in my life.

I thank my readers Ethan Zuckerman and Deb Roy for their patience and extremely helpful feedback. Writing turns out much better while imagining a massively intelligent audience.

Lastly, a shout-out to the current and former DIVE team, Vikas Velagapudi, Yachi Judy Chang, Guru Mahendran and Mike Wang. It's been a pleasure working with you all at some point or another, let's keep hustling to make DIVE a real thing.

Contents

Chapter 1

Introduction

"What we call the past is built on bits."

- John Archibald Wheeler

In the past century, computers have transformed how we think of ourselves, interact with others, design institutions, and conduct scientific work.[1, 2] More recently, computers have also become ubiquitous data collection and storage devices, resulting in an explosion of data describing the state of diverse systems, from international trade networks to subatomic particle interactions.[3]. To explore and understand these systems, scholars have developed a large number of data analysis and visualization techniques that they use to uncover the structure of these systems and to test theories about the dynamics that drive them. Yet, as the data in these systems become increasingly large, our ability to explore them is limited by the absence of more flexible software tools.

There are many software tools and accompanying workflows for data visualization and analysis. But many existing tools present users with a list of possible visualization types and force users to choose a type and specify how their data maps to these types. But what if we inverted this approach, by starting from a user's data and then showing how it could be visualized? **In a sense, a data model should enumerate the whole space of possible visualizations or analyses, and a user's domain-specific knowledge constrains or searches that space. The "relevant" visualizations and analyses are a subset of computationally possible visualizations and analyses.** This is because there are a limited number of dimensions that can be represented meaningfully on a screen, and a limited number of ways that data can be manipulated into those dimensions.

For example, if we had a three column table of people, their occupations, and their incomes, we can easily list the visualizations we could imagine of this dataset. We can bin on occupations to receive a histogram visualizing the number of people in occupations, and we can bin on incomes to receive histogram of the number of people in different income brackets. We can take two columns by grouping occupations to find average incomes, the standard deviation of incomes, and the minimum or maximum of incomes. We can take three columns to form a network connecting people to occupations to income brackets.

In this thesis, we begin by characterizing tools and prior visualization research. Then, we identify common limitations of existing tools. From there, we propose a framework that overcomes these limitations. Lastly, we describe DIVE, a tool implementing this framework. We aim for public release of DIVE by the end of 2015.

1.1 The Data Pipeline

Deriving conclusions from data consists of sequential processes that I call the data pipeline. The data pipeline, and accompanying workflow, usually has four steps: first, collecting data (e.g. experimental observations); second, processing data into a workable form; third, generating analyses or visualizations; finally, integrating analyses and visualizations into a coherent whole (e.g. a paper, article, book). We can unpack the processing step into three more stages: cleaning data, possibly transforming it into an intermediate form, then constructing a data model it in a given system. This pipeline is shown in Figure 3-1, read from left-to-right.

But working with data is rarely truly this straightforward, with feedback loops, multiple agents, and different tools at different steps. For example, a visualization can reveal data errors that must be addressed in the previous data cleaning stage, or a statistical test can indicate that a specific visualization may be meaningful. Further, each stage in the pipeline can be a bottleneck: users can be "stuck" at different points, and these bottlenecks vary depending on circumstances and skillsets.[4]

We are concerned mostly with the last two stages of modeling data and then using it. While many applications and toolsets separate these concerns, or completely hide the data modeling stage, a user's data model should inform his visualizations and analyses.

1.2 Related Work

The past three decades have witnessed a huge body of work in data visualization and analysis, both in academic literature and embodied in tools and libraries.

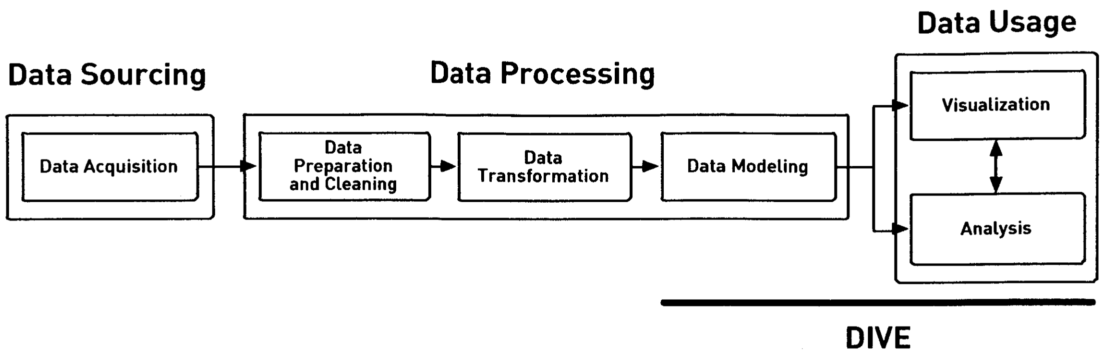


Figure 1-1: General dataflow and conceptual workflow for data-driven projects.

1.2.1 Research

Research advances fall into five domains: development of novel visualization techniques, evaluation of visualization effectiveness, creation of new visualization paradigms and principles, study of workflows, and design of new software architectures and design patterns for building visualizations

We're mainly concerned with the latter two categories. Novel visualization techniques, like using 3D graphics engines to render visualizations, using wearables as visualization interfaces, or manifesting visualizations in physical space, are outside the scope of this thesis, which is concerned more with *what* is presented than *how* it is presented, and therefore uses existing libraries for rendering visualizations. Work in visualization paradigms and principles include research in creating visualization taxonomies[5, 6], visualization principles, [7, 8, 9]. This body of work, lead by Tufte, Card, and Mackinlay, provides heuristics and best-practices, and are invaluable for thinking about single-visualization design. However, they provide few concrete, generative procedures.

Examples of research into new paradigms for visualization tools include InfoVis[10, 11], a toolkit providing data structures and visualization components for Java swing applications; The Information Visualizer[12], a system implementing a user interface concept called the *information workspace* designed around optimizing the amount of user-system interactions; Prefuse[13], a framework providing abstractions for different components in the data pipeline, like I/O libraries, filters, and renderers; Polaris (precursor to Tableau), an tool extending the pivot table interface for exploring multidimensional databases by mapping from relational queries to visualization specifications [14]; DEVise[15], a data exploration system integrating multiple datasets and allowing users to explore visualizations at multiple levels of detail. Some research is not accompanied with an instantiation but is still incredibly informative, like Chi's work on the data state model, a formalism describing the visualization process using a graph model.[16]

These previous visualization frameworks-turned-tools, with early work on visualization languages and automation like SeeDB[17, 18, 19], Visualization Languages [20, 21], Tableau's analytic data engine,[22] and Data Visualization Management Systems [23] are the direct inspiration for DIVE.

1.2.2 Tools

In the data visualization and analysis space, many research papers are usually accompanied by tools, as listed in the previous section. On the other end of the spectrum are software tools and libraries that are necessarily useful but not necessarily based on any framework. A popular subset of these tools and libraries are compiled in Table 1-2.

We can categorize data tools along several dimensions: how accessible the tool is, what in parts of the data pipeline the tool is used, whether the tool is stand-alone, whether the tool is web-based or desktop-based or necessarily used in a development environment, whether the tool is open-source or proprietary, whether the tool is free or commercial, the popularity of the tool, and the intended audience of the tool.

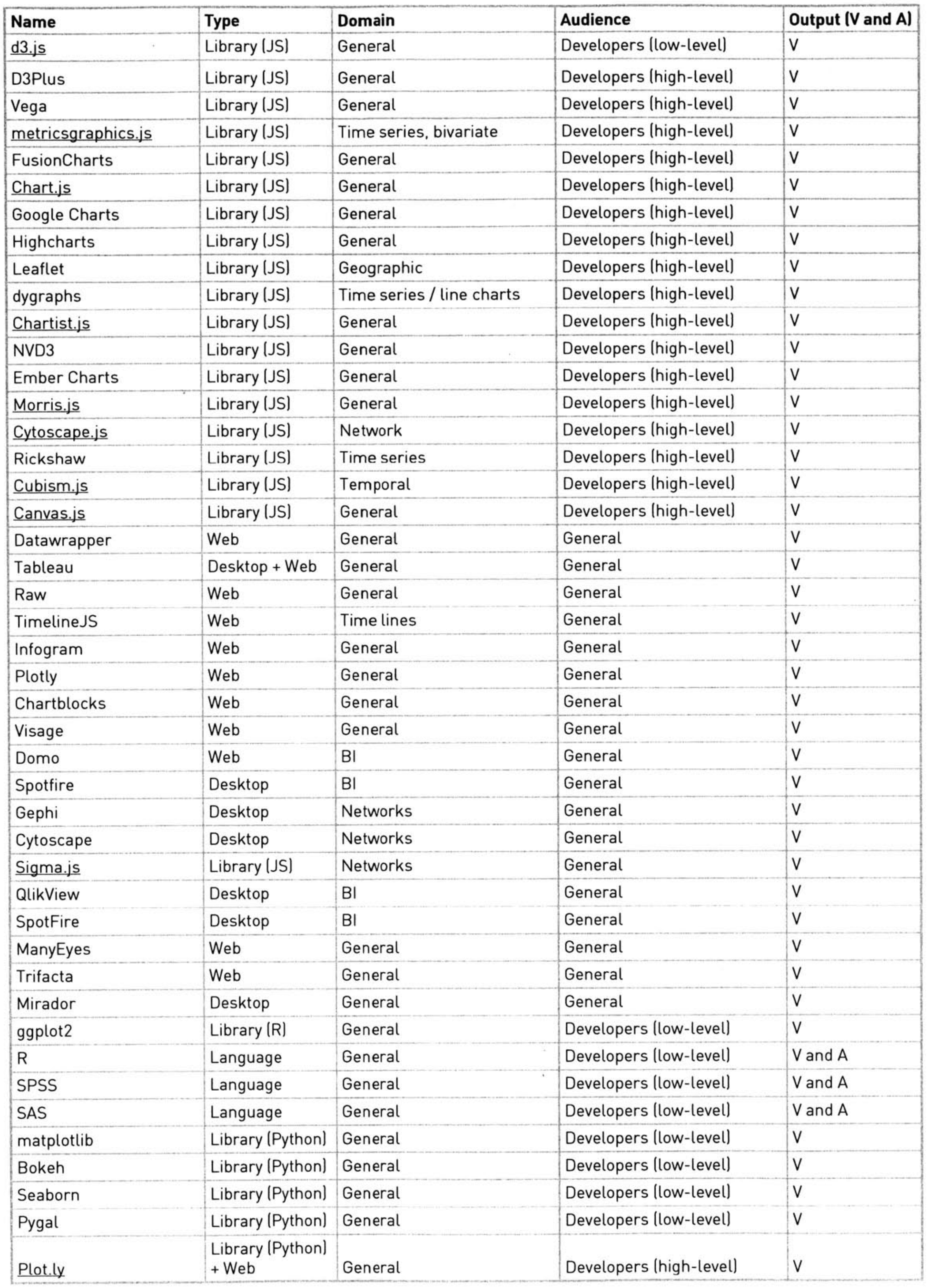
Some open-source tools are libraries accessed by existing languages. For example, Matplotlib is used with python, ggplot with R, Processing, or d3 with Javascript. These libraries are in contrast with stand-alone products. Some of these products are designed specifically for data visualization, like Tableau, DataHero and ManyEyes. Others products, like R, Excel, and MATLAB, are designed for more general data work but include data visualization functionality. Similar exploratory data analysis tools include Mirador, DataHero, and Raw, but approach data visualization from the same outcome-first perspective as previous tools.

1.3 Problem Statement and Motivation

The previously existing tools, which I call *outcome-centric* tools approach data visualization and analysis from the perspective of the outcome. These outcome-centric tools offer a series of plots or analyses to be actively created given a users understanding of their data and the tool. For example, an Excel user can choose to create a piechart or linechart or barchart by directly mapping fields of data to visual attributes.

DIVE inverts the outcome-centric approach by centering on a single idea: there are a finite number of combinations of a dataset, and therefore there are a finite number of common visualizations or analyses of a dataset. Further, with knowledge of the properties of specific datasets and the user's data model, we can filter and rank different visualizations or analyses. In that way, DIVE is a *data-first* tool that, given an understanding of a users data, automatically generating visualizations that can be chosen rather than necessarily created from scratch. In generating these visualizations, in addition to possibly making a user's experience much richer and exploratory, we can also enforce best practices and optimizations.

The goals of this thesis are to show the assumptions, ideas and language that inform our team while we build a tool towards this vision, the implementation details in building this tool, preliminary results about UI design, workflow, analysis, system architecture, and the evaluations we've conducted of our tool thus far. Further, this thesis aims to be useful for the development team working on this project, and our collaborators. Hopefully it can also be useful for future users and other tool builders.



Name	Type	Domain	Audience	Output (V and A)
d3.js	Library (JS)	General	Developers (low-level)	V
D3Plus	Library (JS)	General	Developers (high-level)	V
Vega	Library (JS)	General	Developers (high-level)	V
metricsgraphics.js	Library (JS)	Time series, bivariate	Developers (high-level)	V
FusionCharts	Library (JS)	General	Developers (high-level)	V
Chart.js	Library (JS)	General	Developers (high-level)	V
Google Charts	Library (JS)	General	Developers (high-level)	V
Highcharts	Library (JS)	General	Developers (high-level)	V
Leaflet	Library (JS)	Geographic	Developers (high-level)	V
dygraphs	Library (JS)	Time series / line charts	Developers (high-level)	V
Chartist.js	Library (JS)	General	Developers (high-level)	V
NVD3	Library (JS)	General	Developers (high-level)	V
Ember Charts	Library (JS)	General	Developers (high-level)	V
Morris.js	Library (JS)	General	Developers (high-level)	V
Cytoscape.js	Library (JS)	Network	Developers (high-level)	V
Rickshaw	Library (JS)	Time series	Developers (high-level)	V
Cubism.js	Library (JS)	Temporal	Developers (high-level)	V
Canvas.js	Library (JS)	General	Developers (high-level)	V
Datawrapper	Web	General	General	V
Tableau	Desktop + Web	General	General	V
Raw	Web	General	General	V
TimelineJS	Web	Time lines	General	V
Infogram	Web	General	General	V
Plotly	Web	General	General	V
Chartblocks	Web	General	General	V
Visage	Web	General	General	V
Domo	Web	BI	General	V
Spotfire	Desktop	BI	General	V
Gephi	Desktop	Networks	General	V
Cytoscape	Desktop	Networks	General	V
Sigma.js	Library (JS)	Networks	General	V
QlikView	Desktop	BI	General	V
SpotFire	Desktop	BI	General	V
ManyEyes	Web	General	General	V
Trifacta	Web	General	General	V
Mirador	Desktop	General	General	V
ggplot2	Library (R)	General	Developers (low-level)	V
R	Language	General	Developers (low-level)	V and A
SPSS	Language	General	Developers (low-level)	V and A
SAS	Language	General	Developers (low-level)	V and A
matplotlib	Library (Python)	General	Developers (low-level)	V
Bokeh	Library (Python)	General	Developers (low-level)	V
Seaborn	Library (Python)	General	Developers (low-level)	V
Pygal	Library (Python)	General	Developers (low-level)	V
Plot.ly	Library (Python) + Web	General	Developers (high-level)	V

Figure 1-2: Categorized collection of data visualization and analysis tools and libraries.

Chapter 2

Conceptual

Why is it so simple to create a chart-builder tool for building scatterplots given two columns in a dataset, but so difficult to build a tool that is a little bit smarter? Because with the huge varieties of data structure, size, and visualization types that exist, the smallest bit of flexibility requires a huge amount additional complexity.

One approach is to limit our scope. We dont deal with all data structures, but with those that are common and easy to reason with. The same holds for visualizations, in that we wont deal with all visualization types, but with those that visualization creators are used to reasoning with, and that visualization consumers are used to interpreting. Or, with statistical analyses, we will be concerned with the simplest statistical tests and models before proceeding to more complex tests and models.

In any case, we start by defining the entities we are dealing with and establishing rules for how they are dealt with. Then, we can attempt to create a framework with which we can think about the relationship between data, visualization, and analysis, and finally state some of the ideas and procedures that are core to DIVE. We start by establishing a language that exactly describes data sets, operations on datasets, and achievable data structures from chaining operations. This language will draw upon notation from set theory and computer science. While it will be clumsy at times, the formalism tries to both capture common intuitions about manipulating and describing data, but also exactly describe these operations. Sometimes an operation, like determining hierarchical structure, would be most easily represented with pseudocode. Other times, like in describing data structure, set notation is most appropriate.

Then, we will abstracting the concept of datasets into a tractable form and enumerate visualizable and analyzable data structure for given datasets. Finally, we will map from these data structures to specifications for visualizations and analyses. The conceptual progression from data sources to final specifications of visualizations and analyses is shown in Figure 2-1, progressing from left-to-right. Each stage is a gray rectangle, and the important data within each stage is a nested white rectangle.

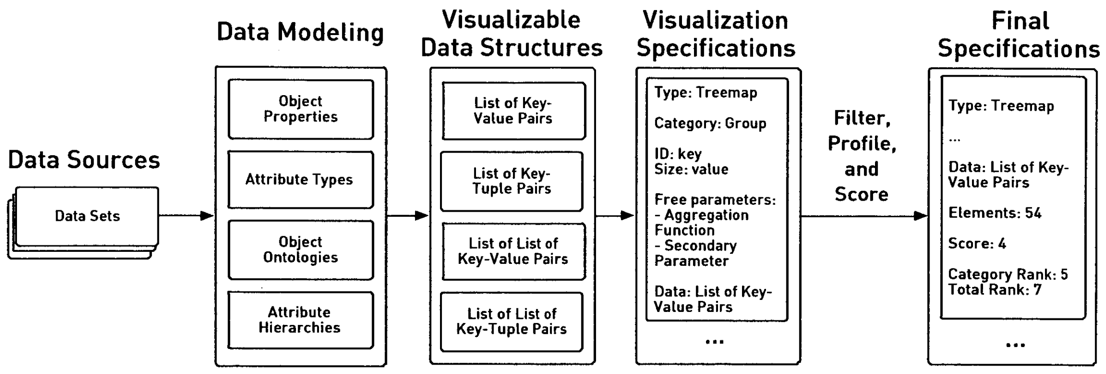


Figure 2-1: Data State Diagram of our proposed Specification Generation Pipeline.

One measure of success is if this formalism can produce the exact set of optimal visualizable data structures given an arbitrary dataset. I'd say that false positives are acceptable; and that more important is creating a framework accommodating to user input. At the least, we'd want a framework giving visualization creators a path forward to thinking rigorously about mapping from data to visualizations.

2.1 Brief Description of Visualization

Visualization used in diverse fields, from journalism to scientific research to design, with practitioners coming from equally diverse casts of mind. We have designers creating beautiful visualizations for public consumption, developers building visualization to monitor system state, scientists creating visualizations to reason about and communicate experimental data, and so on. Visualization is one medium for representing, reasoning with, and communicating complexity.

Here we are concerned with 2D visualizations created and consumed on a flat surface, usually a screen or on paper. We're concerned specifically with data visualizations, in which the state, relative orientation, and relative position of visual elements are determined by the value of quantitative variables. Unfortunately, not all visual dimensions are created equal. Look at the visualization below in Figure 2-2.

Here, volume is used to encode dimension. However, both the absolute and relative magnitude of these 3-D disks is hard for a user to estimate. In this case, using just a list would have worked.

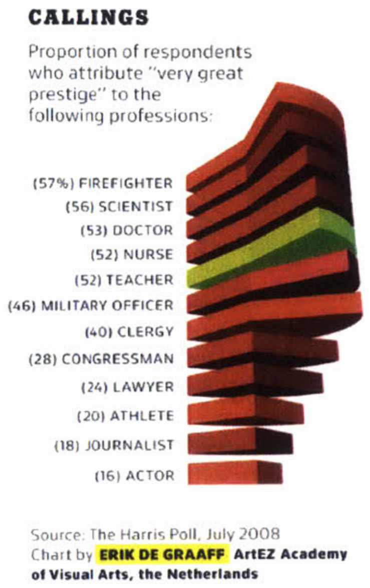


Figure 2-2: A three dimensional, layered take on the pie chart: visualization of survey data measuring the amount of prestige attached to specific professions.

The visualization in Figure 2-3 has better use of visual dimensions than the previous pie chart in Figure 2-2. It encodes *more* information than Figure 2-2, and even includes redundancies (both size and Y position encode company value) but it is clear. Perhaps this isn't the provably optimal visualization of this data, but let's start from the following statements: 1) the fewer visual dimensions used to represent data, the better; and 2) some visual dimensions are better[1] [24] for encoding data than other dimensions. The risk is not only that poor visualizations are difficult to interpret, but that they are misleading.

2.2 Dimensions of Semiotic Space and Visualization Types

Some visual dimensions are experimentally shown to be better than otherscan be quantitatively shown not created equal. In particular, there are seven visual dimensions we are concerned with: 1. position (X and Y), 2. length, 3. area, 4. volume, 5. value, 6. color hue, 7. orientation, and 8. shape.[25]

Common visualization types are just conventional uses of these visual dimensions. The simplest scatterplot uses X and Y position to encode data, a treemap uses area but forgoes the exact use of position, a geographic map uses value or color but forgoes position, length, and area, and so on. In

[1]"Better" meaning that the relative magnitude of elements in some dimensions are more accurately estimated, that changes in these dimensions are more readily detected, that they facilitate classification, and take advantage of pre-attentive features.

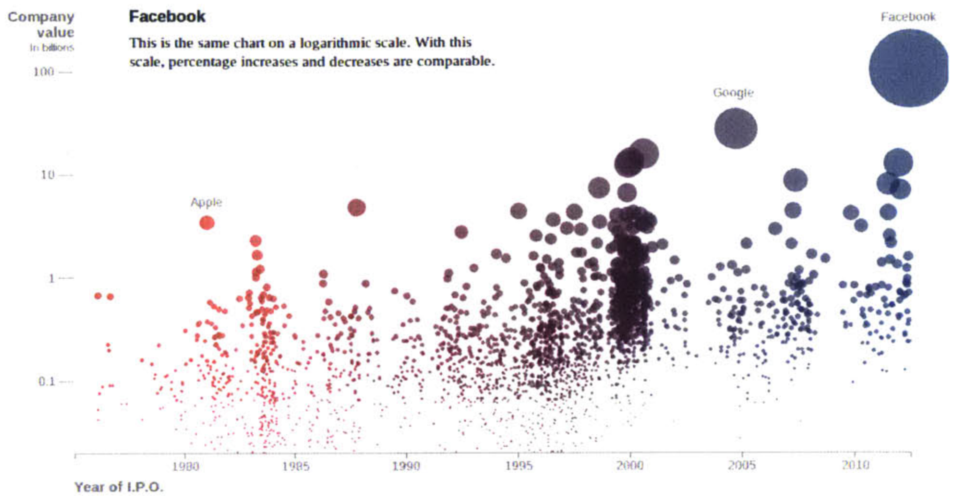


Figure 2-3: A scatterplot showing the relation between year of I.P.O. and company value.

this way, every visualization type is a "formula" that assembles bits into pixels.

We are going to constrain the problem by dealing with visualizations that possess, at most, four "free" dimensions. By free dimensions, we mean dimensions that are not structurally part of the visualizations, like country position and size on a standard world map. In particular we are dealing with:

2.3 Data Structures for Visualization

What is the minimum amount of structure we need in data for the data to be visualizable? What is the maximum amount of structure in data that we can reasonably and possibly visualize? Here, we mean data structure not in the strict computer science sense (i.e. of particular organizational schemes of data that possess certain operational qualities) but in the sense of different kinds of collections of data.

The most general structure dataset is a list of points of with arbitrary amounts of attached data, represented as a list of length L containing tuples of length n:

$$[(v_{11}, v_{12}, ..., v_{1n}), (v_{21}, v_{22}, ..., v_{2n}), (v_{L1}, v_{L2}, ..., v_{Ln})]$$

As shorthand, we can denote this same structure as $[(v_n)]_L$, which captures the cardinality L and dimension n. Because this structure is flat, and our our data can be more easily interpreted if it is more nested, we see the following structures more often:

Visualization Type	Free Dimensions	Num. Dimensions
Scatterplot	X, Y, size, color	4
Treemap	Size, color	2
Piechart	Size, color	2
Linechart	X, height, color	3
Barchart	X, Y, color	3
Histogram	X, Y	2
Network	Node size, link size, node color, link color	4
Parallel coordinates	X, Y, color	3
Geographic (discrete)	Color	1
Geographic (continuous)	X, Y, Size, Color	2

Figure 2-4: Table of visualizations and associated visual dimensions that we are concerned with (not comprehensive but covers common use cases).

1. List of key-value pairs: $L_{k,v} = [\{k : v\}]_L$

   This data structure could be represented many as a treemap (size), scatterplot (x, y), barchart (x, y), network (edge existence between nodes k and v), and geometric maps (if k is a geographic entity).

2. List of list of key-value pairs: $L_{k,L_{j,v}} = [\{k : [\{j : v\}]_{L_j}\}]_{L_k} = [\{k : L_{k,v}\}]_{L_k}$

   This data structure is most commonly used to represent multiple line charts, or time series. It is also used to represent groupings of elements in grouped visualizations like treemaps. It is also used to generate multiple grouping visualizations, akin to faceting.

3. List of key-tuple pairs: $L_{k,\vec{v}} = [\{k : (v_1, v_2, ..., v_n)\}]_{L_k} = [\{k : (v_n)\}]_{L_k}$

   This data structure is commonly used to attach more data to the visualizations generated by structure 1) $L_{k,v}$.

4. List of list of key-tuple pairs: $L_{k,L_{j,\vec{v}}} = [\{k : [\{j : (v_n)\}]_{L_j}\}]_{L_k} = [\{k : L_{k,\vec{v}}\}]_{L_k}$

   This high-dimensional dataset is most often used to encode groupings of time series or line charts, or multiple time series or line chart plots.

A key factor approach to determining the mapping from datasets to these data structures is dimensional analysis and keeping track of cardinalities. Since these four data structures can all be mapped to a list of tuples, we can count the dimensions of the list of tuples that encodes the same information as these data structures to get their effective dimensionality.

Number	Data structure	Dimension
1	$L_{k,v} = [\{k : v\}]_L$	$L \times 2$
2	$L_{k,L_{j,v}} = [\{k : [j : v]\}]_L = [k : L_{k,v}]_L$	$L_k \times L_j \times 2 = 3$
3	$L_{k,\vec{v}} = [\{k : (v_1, v_2, ..., v_n)\}]_L$	$L \times n = n + 1$
4	$L_{k,L_{j,\vec{v}}} = [\{k : [\{j : (v_1, v_2, ..., v_n)\}]\}]_L$	$L_k \times L_j \times n = n + 2$

2.4 Describing Datasets

The three simplest conceptual types of datasets:[26]

- Long datasets. The record model of datasets, in which each row is an instance of an object with attributes described by the columns.
- Wide datasets. Each row contains an instance of a series of values. Pivoting is the operation of transforming a long dataset into a wide dataset.
- Network datasets. Network datasets can be either long or wide (edge-list or incidence matrix), with accompanying node and edge data.

In all three of these cases, the concept of an object is key: what data is an object, what is an instance, and what is an attribute of an object? To give examples that we will return to later, in a long dataset each row can be thought of as an instance of an object. If the long dataset has headers, these headers can be taken as attributes of that object. Network datasets can be expressed as long and wide datasets. Wide datasets can be unpivoted to long datasets as necessary.

2.4.1 Types: Categorical vs. Quantitative

Our model of data types follows the classification of variables in statistics. That is, every variable is either categorical (C) or quantitative (Q). Categorical variables, also called discrete or qualitative variables, are meaningfully countable, cannot be and only take on a certain set of values. Categorical variables include binary variables. Examples of categorical variables are gender, names, and cities. In contrast, quantitative, or continuous, variables are not countable.

However, certain mathematical operations, such as addition and multiplication, are meaningful for quantitative but not categorical variables. Quantitative variables include count variables and real-valued variables. Examples of quantitative variables include age, height, and income. Note that the distinction is based on interpretation rather than data type; while strings are always categorical, numeric variables are not necessarily quantitative (e.g. numeric, discrete survey entries are categorical).

Temporal variables are a special case because time can be considered both categorical and quantitative, as seen in Tableau's type system.[19] Like many quantitative variables, temporal variables have a natural ordering and has well-defined ratios. But unlike quantitative variables, certain temporal variables can also be treated as categorical variables, like month or hour. Further, there are analysis techniques (like forecasting and de-seasonalization in time series analysis) that are valid for all numeric domains but are only meaningfully interpreted with a temporal variable. However, in our case, temporal variables are not distinct from categorical or quantitative variables and can be either depending on user input.

Type Detection

Automatically detecting the type of an arbitrary list $\vec{X}$ is not always reliable and ultimately relies on user corrections. But our type detection system are based on computer data types and is based on a combination of methods: 1) detecting predefined attribute labels (e.g. labels like "Country" or "Year" are heavily weighted), 2) random sampling from X and checking elements against predefined sets, 3) regular expression matching from most general types (e.g. numeric) to most specific (e.g. float).

Relating these computer data types to statistical data types, we classify strings, geographic entities, booleans as categorical. Integers and floats are quantitative. Lastly, datetimes in various formats (e.g. UTC, ISO) names of time periods (e.g. week, days, months) are categorical.

2.4.2 Transformation and Aggregation Functions

Here we will define general types of functions that operate on our datasets, categorized by their inputs, effect on dimension, and effect on cardinality.

Functions on list a of N elements

Functions acting on lists of cardinality N either preserve dimension or reduce dimension, and can return either a single list as a result or a tuple of lists. These families of quantitative transformation functions are defined as $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$, transformation $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that $m \leqslant n$, and aggregation $A : \mathbb{R}^n \rightarrow \mathbb{R}^1$. F denotes functions that preserve cardinality, such as cumulative sum and cumulative average. T denotes functions that reduce cardinality, but not to a scalar value, and includes term-to-term difference and calculating quantiles. A denotes functions that reduce lists to scalar values, mostly used for descriptive statistics, like maximum, minimum, mean, and standard deviation.

Functions on multiple lists of N elements

Here, standard notation for dimensionality gets clunky, but there are really one case we're concerned with: functions that apply operations pair-wise across input lists of the same cardinality (e.g. summing two rows) and output a list of the same cardinality as the inputs. We will denote this function as $T_{N_Q \to 1}$.

2.4.3 Entities

Core to building an object model of our datasets is finding the fields that denote entities. While this too depends on user input, we can attempt to detect entities by 1) detecting uniqueness of categorical fields and 2) finding the similarity between categorical fields. In case two we can define a similarity function $s(\vec{x}, \vec{y})$ that accepts two vectors of equal length. If $s(\vec{x}, \vec{y}) \geqslant C$, then we say that $\vec{x}$ and $\vec{y}$ represent the same entity $\vec{e} = \vec{x}_{\neq} \cup \vec{y}_{\neq}$ such that $\vec{x}, \vec{y} \subset \vec{e}$

2.4.4 Hierarchical Relationships

Define ordered hierarchical tuple of categorical vectors $\mathcal{H} = \left(\vec{C}_i, ..., \vec{C}_N\right)$. For two vectors $\vec{C}_i$ and $\vec{C}_j$, with $1 \leqslant i < j \leqslant N$, and the corresponding ordered unique vectors $\vec{C}_{i\neq}$ and $\vec{C}_{j\neq}$, the following relation holds: For any two elements $c_{i,n}, c_{i,m} \in \vec{C}_{i,\neq}$, with $n \neq m$, the sets of corresponding values $c_j(C_i = c_{i,n})$ and $c_j(C_i = c_{i,m})$ from $\vec{C}_{j,\neq}$ are distinct. That is, $c_j(C_i = c_{i,n}) \cap c_j(C_i = c_{i,m}) = \varnothing$. Further, we require that $|\vec{C}_{i/ne}| < |\vec{C}_{j/ne}|$, making the assumption that the parents in a hierarchical relationship have less elements than children.

In practice, due to either messy data or duplicate names, we can define an overlap function $O(x, y)$ and a constant C such that if $O(c_j(C_i = c_{i,n}), c_j(C_i =, c_{i,m})) \leqslant C$ we can effectively state that $c_j(C_i = c_{i,n})$ and $c_j(C_i = c_{i,m})$ are distinct. For example we can use the Jaccard coefficient as the overlap function $O(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$.

Consider the following table of unique cities and the countries and continents in which they belong:

An example of this procedure is the following: The set of the first Continent column is $\vec{C}_{1\neq} =$ {North America, Africa, Asia}. Each element in this list has the corresponding Country sets $\vec{C}_2(C_1 =$ North America$) =$ {Canada, USA}, $\vec{C}_2(C_1 =$ Africa$) =$ {Nigeria, Morocco}, and $\vec{C}_2(C_1 =$ Asia$) =$ {India, China, Japan}.

Continent	Country	City
North America	Canada	Toronto
North America	Canada	Vancouver
North America	USA	Houston
Africa	Nigeria	Lagos
Africa	Nigeria	Abuja
Africa	Morocco	Casablanca
Asia	India	New Delhi
Asia	China	Shanghai
Asia	China	Beijing
Asia	Japan	Kyoto

Table 2.1: Table of cities, with associated country and continents

2.5 Mapping from Datasets to Visualizable Data Structures

We start with a dataset with an arbitrary amount of categorical and quantitative fields:

C_1	...	C_{N_C}	Q_1	...	Q_{N_Q}
$c_{1,1}$	...	c_{1,N_C}	$q_{1,1}$	...	q_{t,N_Q}
$c_{1,1}$	...	c_{1,N_C}	$q_{2,1}$	...	q_{t,N_Q}
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$c_{1,1}$	...	c_{1,N_C}	$q_{n,1}$	...	q_{t,N_Q}

Starting with some notation, $N_C = |\mathcal{C}|$, and $N_Q = |\mathcal{Q}|$. We define the variable $\mathcal{D} = \{C_1, ...C_{N_C}, Q_1, ..., Q_{N_Q}\} = \{\mathcal{C}, \mathcal{Q}\}$ where $\mathcal{C} = \{C_1, ...C_{N_C}\}$ and $\mathcal{Q} = \{Q_1, ...Q_{N_Q}\}$. Given any vector $\vec{X}$, the vector containing the distinct elements of $\vec{X}$ can be notated as $\vec{X}_{\neq}$

With a our distinction between categorical and quantitative variables, by counting the number of both types in a dataset, we can classify a dataset into one of eight cases, as shown in Table 2.2

	$N_C = 0$	$N_C = 1$	$N_C > 1$
$N_Q = 0$	-	C	F
$N_Q = 1$	A	D	G
$N_Q > 1$	B	E	H

Table 2.2: The eight cases used to categorize datasets based on the number of categorical and quantitative fields.

2.5.1 Cases A, B: Only quantitative variables

In the first two cases we deal with a purely quantitative dataset with no categorical variables. On one hand, numeric datasets lend themselves to easy visualization *as-is*, in that a visualization creator can plot raw values. On the other hand, numeric datasets can be meaningfully and naturally transformed.

Case A) One quantitative variable and no categorical variables ($N_Q = 1$ and $N_c = 0$)

An example dataset with this structure is shown in Table 2.4, which is a column of daily high temperatures that we will refer to as $\vec{Q}_i$, the elements of which are q_i.

High Temperature
98
80
98
92
⋮
74

Table 2.3: Example of $N_Q = 1$: Daily High Temperature Measurements

Keeping in mind that we need to create data with *at least* as much structure as a list of key-value pairs (Structure 1), we rule out the possibility of plotting the data *as-is* as $[q_i]$, which results in a one dimensional list, or aggregating the data into $A(\vec{Q})$, which results in a single scalar value.

There are four intuitive procedures to map this data into a list of key-value pairs:

1. Plot the temperatures against their respective indexes, so we get data of form $\boxed{[\{\text{index}(\vec{Q}_i, q_i) : q_i\}]}$. In this case, we get [{1: 98}, {2: 80}, ..., {N: 74}], which we can plot as a scatterplot, barchart, or linechart.

2. Count the number of occurrences of each high temperature, to get data of form $\boxed{[\{q_{i\neq} : \text{count}(\vec{Q}_i, q_{i\neq})\}]}$. We can plot this as a treemap, piechart, or barchart.

3. Bin the temperature values into $N_{\mathcal{B}}$ ranges $\mathcal{B} = (b_1, b_2, ..., b_{N_{\mathcal{B}}})$ (e.g. 0-40, 40-80, 80-120), which can be one-sided or two-sided, and inclusive or exclusive. Then, we map each bin range to the a function of the values within that range. For example, we can calculate the number of high temperatures within 80 and 120, or find the average temperature within that range. Here we get data of form $\{b_n : A(q_i \in b_n\} \forall b_n \in \mathcal{B}$, or $\boxed{[\{b_n : A(q_i \in b_n\}]}$, where A is an aggregation function. This can also be represented as a treemap, piechart, or barchart.

4. We can transform the column values into a vector $T(\vec{Q})$ and proceed do either of the three steps above. For instance, we can calculate the cumulative average and then plot each element against respective indexes as a scatterplot.

High Temperature
98
80
98
92
⋮
74

Table 2.4: Example of $N_Q = 1$: Daily High Temperature Measurements

Case B) Multiple quantitative variables and no categorical variables ($N_Q > 1$ and $N_c = 0$)

Here we have a set of N_Q quantitative vectors $\mathcal{Q} = \{\vec{Q}_1, \vec{Q}_2, ..., \vec{Q}_{N_Q}\}$. An example dataset fitting this case is shown in Table 2.5, which has two columns of low temperature and humidity appended to the previous Table 2.4. Since we have N_Q single quantitative vectors, this case contains N_Q instances of Case A.

High Temperature	Low Temperature	Humidity
98	72	10
80	65	14
98	82	12
92	70	9
⋮	⋮	⋮
74	60	23

Table 2.5: Example of $N_Q = 3$: High temperature, low temperature, and humidity measurements

But unlike Case A we can now create a derived column based on function of the N_Q vectors, denoted as $\vec{Q}' = T_{N \to M}(\mathcal{Q})$. For example, the function $f(\mathcal{Q}) = \vec{Q}_1 - \vec{Q}_2$ will find the pairwise difference difference between the high and low temperatures. From then, we have another case that fits into Case A. The number of derived columns we compute is denoted as $N_{Q'}$. Given these derived columns, now have two actions that can act on multiple vectors of numeric data, and thus two new ways to generate 2-D key-value pairs of form $[\{k : v\}]$:

1. We can take the raw data *as-is* and plot the elements of one vector against the corresponding elements of another to create data of the form $[\{q_i : q_j\}] = \{q_i : q_j\}]$. There are $\binom{N_Q + N_{Q'}}{2}$

possibilities of this case where N_{V_0} denotes the number of derived columns. For example. we can plot humidity against temperature range as a scatterplot.

2. We can bin across one variable and aggregate on the corresponding values of another. So if we have a vector $\vec{Q}_i$ binned into intervals $b_{n,\vec{Q}_i}$, the corresponding values of another vector $\vec{Q}_j$ are $A(q_j(q_i \in b_n)$. Then, our data structure is of form $\boxed{[\{b_{n,\vec{Q}_i} : A(q_j(q_i \in b_n)\}]}$. For Table 2.5, an example is finding the average humidity for high temperatures between 60 and 100. This can be plotted as a barchart or scatterplot.

Further, we can also create lists of tuples $[\{k : (v_n)\}]_{L_k}$. There are two methods to generate this structure, analogous to the previous procedures. We can:

1. We can take the raw data *as-is* and plot the elements of one vector against multiple corresponding elements of another to create data of the form $\boxed{[\{q_i : (q_j, ..., q_{N_Q}\}]}$. For example, we can use a scatterplot to plot high temperature (X) against low temperature (Y) and include the humidity (size) of each record.

2. We can bin across one variable and aggregate on the corresponding values of multiple other variables. Then, our data structure is of form $\boxed{[\{b_{n,\vec{Q}_i} : (A(q_j(q_i \in b_n), ..., A(q_{N_Q}(q_i \in b_n))]\}]}$. An example is finding the highest humidity and highest low temperature for different intervals of high temperatures, representable as a barchart with color encoding a dimension.

2.5.2 Cases C, D, E: One categorical variable

With a categorical variables, we now have the notion of performing aggregations not on bins but on elements. For example, we can find the average income for teachers vs firefighters. Aggregation is not meaningful when all elements are unique, but this is a specific case that will be excluded from the general inferred data structures.

Case C) One categorical variable and no quantitative variables ($N_C = 1$ and $N_Q = 0$)

An example dataset with this structure is shown in Table 2.6. With one categorical variable $\vec{C}$ and no numeric variables, it is only possible to form a visualizable data structure by counting the unique values of $\vec{C}$ to receive data of structure $\boxed{[\{c_{i\neq} : \text{count}(\vec{C}, c_{i\neq}\}]}$. In our example, procedure creates a treemap or barchart using size to show the number of reviews a book has received. This counting is meaningless for unique elements, and should not be performed.

Amazon Reviews
The Blank Slate
Sapiens
Why Information Grows
Rewire
Alone Together
Alone Together
1Q84 ⋮
Sputnik Sweetheart

Table 2.6: Example of $N_C = 1$: A list of products reviewed on Amazon, with one entry per review.

Case D) One categorical variable and one quantitative variable ($N_C = 1$ and $N_Q = 1$)

We now append one column to the previous book review data to include the number of stars in the review. These two columns we denote as the categorical variable $\vec{C}$ and the numeric variable $\vec{V}$. This case contains one instance of Case A and one instance of Case C.

Amazon Reviews	Review Stars
The Blank Slate	5
Sapiens	4
Why Information Grows	5
Rewire	5
Alone Together	4
Alone Together	3
1Q84	4
⋮	⋮
Sputnik Sweetheart	5

Table 2.7: Example of $N_C = 1$ and $N_Q = 1$: A list of products reviewed on Amazon and the number of stars in review, with one entry per review.

There are two procedures to map this data into a list of key-value pairs:

1. If the categorical variable is unique, plot the values of the categorical variable against the corresponding quantitative variable to get $\boxed{[\{c_i : v_i\}]}$. This can be represented as a treemap with size corresponding to review, or barchart with height corresponding to review.

2. We can aggregate on $\vec{C}$ by a function A of $\vec{V}(c_{i\neq} | c_{i\neq} \in \vec{C_{\neq}})$, we receive data of structure $\boxed{[\{c_{i\neq} : \mathcal{A}(\vec{V}(c_{i\neq}))\} \; \forall \; c_{i\neq} \in \vec{C_{\neq}}]}$, where $c_{i\neq}$ are the unique elements of $\vec{C}$.

Case E) One categorical variable and multiple quantitative variables $N_C = 1$ and $N_Q > 1$

Here, we expand to the general case of an multiple quantitative variables by including the date of a review in our dataset, shown in Table 2.8. In the general case of $N_Q > 1$, just like in Case

C, can we perform arbitrary functions on the set of continuous vectors $\mathcal{Q} = \{\vec{Q}_1, \vec{Q}_2, ..., \vec{Q}_{N_Q}\}$. If $N_Q = 2$, for instance, we can perform a pair-wise functions on two continuous variables to produce a derived column $\vec{Q}' = \mathcal{T}_{N \to M}(\vec{Q}_1, \vec{Q}_2)$. As a function of all continuous vectors, we have a function accepting N_Q inputs $\vec{Q}' = \mathcal{T}_{N \to M}(\vec{Q}_1, \vec{Q}_2, ..., \vec{Q}_{N_Q})$. With a categorical variable, now we can perform aggregations on $\vec{Q}_0$ like in the case of a single numeric variable. Here, we receive data of structure $\boxed{[\{c_{i\neq} : \mathcal{A}(\vec{Q}'(c_{i\neq}))\} \ \forall \ c_{i\neq} \in \vec{C}_{\neq}]}$.

Amazon Reviews	Review Stars	Date
The Blank Slate	5	8/1/15
Sapiens	4	6/2/15
Why Information Grows	5	7/20/15
Rewire	5	7/10/14
Alone Together	4	6/3/15
Alone Together	3	1/2/14
1Q84	4	8/7/15
⋮	⋮	⋮
Sputnik Sweetheart	5	8/7/15

Table 2.8: Example of $N_C = 1$ and $N_Q = 1$: A list of products reviewed on Amazon and the number of stars in review, with one entry per review.

This case contains contains N_Q instances of Case C, one instance of case B, and N_Q instances of case D.

2.5.3 Cases F, G, H: More than one categorical variable

Here we have a set of categorical vectors $\mathcal{C} = \{\vec{C}_1, \vec{C}_2, ..., \vec{C}_{N_C}\}$, a set of entities $\mathcal{E} = \{\vec{E}_1, \vec{E}_2, ..., \vec{E}_{N_E}\}$, and a set of hierarchical relationships $\mathcal{H}$. With more than one categorical variable, we introduce data structures that represent connections between categorical variables and can be visualized as networks. In Cases F-H, we are considering those in which every categorical variable represents a separate entity, so all visualized networks are N-partite networks. That is, all connections are between nodes of separate types. The case of multiple categorical variables with an unequal number of entities occurs in two instances: first, the dataset has a hierarchical relationship, second, the dataset has an multiple occurrences of the same entity.

Case F) $N_C > 1$ and $N_Q = 0$

An simple example of this case is a dataset of mentions on Twitter, showing only the source profile and destination profile.

We start with categorical variables $\mathcal{C} = \{\vec{C}_1, \vec{C}_2, ..., \vec{C}_{N_C}\} = \mathcal{E} = \{\vec{E}_1, \vec{E}_2, ..., \vec{E}_{N_E}\}$, and an example given in Table 2.9. We can create data structures taking pairs of raw variables: $\boxed{[\{[e_i : e_j]\}]_{N_E}}$ where $i \neq j$ and $i, j \leqslant N_E$. There are $\binom{N_E}{2}$ of these possibilities. Further, we can send combinations of these pairs, representing an N-partite network $\boxed{[\{(c_i, c_j, c_k)\}]}$

There are N_C cases of B in this case, $[\{c_{i\neq} : \text{COUNT}(\vec{C}, c_{i\neq}\}]$.

Twitter Mention Source	Twitter Mention Destination
@whitehouse	@CIA
@cesifoti	@macroMIT
@kevinzenghu	@cesifoti
⋮	⋮
@MIT	@cesifoti

Table 2.9: Example of $N_C = 2$ and $N_Q = 0$: A list of mentions on Twitter, with only the source and the destination of the mention.

Case G) $N_C > 1$ and $N_Q = 1$

Let's append a numeric column to the previous table showing the number of favorites that a tweet received, as shown in Table 2.10. The new case is that the numeric column is a property of either the entities or the edge (in this case, the number is a column of the edge but one can imagine having a column like "Destination Followers" that is a a property of the destination node).

Given that we are interpreting each categorical variable as a node in a network, a quantitative variable $\vec{Q}$ in this dataset can describe either node attributes or an edge attribute. This depends on the the user's specification, but if $\vec{Q}$ is interpreted as a node attribute, we can create accompanying node data in the form $[\{e_i : q\}]$. If $\vec{q}$ is interpreted as an edge attribute, we can create edge data as $\boxed{[\{(e_i, e_j) : q\}]}$.

Twitter Mention Source	Twitter Mention Destination	Favorites
@whitehouse	@CIA	102
@cesifoti	@macroMIT	32
@kevinzenghu	@cesifoti	5
⋮	⋮	⋮
@MIT	@cesifoti	15

Table 2.10: Example of $N_C = 2$ and $N_Q = 1$: A list of mentions on Twitter, the source, destination, and number of favorites of the mention.

Case H) $N_C > 1$ and $N_Q > 1$

Lastly, with more than one quantitative variable, we have an arbitrary amount of node and edge data, as shown in Table 2.11. That is, we can create $\boxed{[\{(e_i, e_j) : (q_k, ..., q_m)\}]}$. Here we include one instance of Case B, one instance of Case E, and N_Q instances of Case G.

Mention Source	Mention Destination	Favorites	Source Followers	Target Followers
@whitehouse	@CIA	102	6600000	834200
@cesifoti	@macroMIT	32	5795	908
@kevinzenghu	@cesifoti	5	505	5795
⋮	⋮	⋮	⋮	⋮
@MIT	@cesifoti	15	32000	5795

Table 2.11: Example of $N_C = 2$ and $N_Q = 3$: A list of mentions on Twitter, with the source, the destination of the mention, the number of followers of the source, and number of followers of the target.

2.5.4 Summary

In general, we begin with a dataset with attributes of various types. From the simplest cases of single categorical or quantitative attributes, we see that there are a finite number of meaningful transformations that can act on our datasets, and a finite number of visualizable data structures that can come out of the data. With both a categorical and quantitative variable, we can group categorical variables and aggregate quantitative variables based on that grouping.

For the general case of more than one categorical variable, we can have data structures that represent networks, though we need to be mindful of hierarchical structure and repetitions of entities. For the general case of more than one quantitative variable, we can have functions that act on multiple quantitative fields to produce derived fields. The more general cases include several instances of the simpler cases, depending on the number of variables.

2.5.5 Visualization data with time

With time variables, the amount and type of visualizable data structures of a dataset do not change. It depends on whether time is considered a categorical variable or quantitative variable. In our case, unless time is explicitly categorical (for example, in data with "week" as a field), it will always be treated as a quantitative variable that can be binned. However, time does differ from other quantitative variables because of standard bins: we default to binning by predefined periods, like hours, days, weeks, years.

2.6 Conditionals

Conditional statements, also called selections in relational algebra or filters in scientific computing, act on a collection of elements and returns only those elements evaluating to true given a

	$N_C = 0$	$N_C = 1$	$N_C > 1$
$N_Q = 0$	$\varnothing$	C) $[\{c_{i\neq} : \text{count}(\vec{C}, c_{i\neq}\}]$, $[\{c_i : v_i\}]$ if unique, $[\{c_{i\neq} : \text{count}(\vec{C}, c_{i\neq}\}]$ if not unique	F) $\{[e_i : e_j]\}]_{N_E}$, $[\{(c_i, c_j, c_k)\}]$, and N_C cases of B.
$N_Q = 1$	A) $[\{\text{index}(\vec{Q}_i, q_i) : q_i\}]$, $[\{q_{i\neq} : \text{count}(\vec{Q}_i, q_{i\neq})\}]$, $[\{b_n : A(q_i \in b_n\}]$	D) $[\{c_{i\neq} : \mathcal{A}(\vec{V}(c_{i\neq}))\} \forall c_{i\neq} \in \vec{C}_{\neq}]$, one case of C, and one case of A.	G) $[\{(e_i, e_j) : q\}]$, one case of F.
$N_Q > 1$	B) $[\{b_{n,\vec{Q}_i} : A(q_j(q_i \in b_n)\}]$, $[\{q_i : (q_j, ..., q_{N_Q}\}]$, $[\{b_{n,\vec{Q}_i} : (A(q_j(q_i \in b_n), ..., A(q_{N_Q}(q_i \in b_n))$, and N_Q cases of A	E) $[\{c_{i\neq} : \mathcal{A}(\vec{Q}'(c_{i\neq}))\} \ \forall \ c_{i\neq} \in \vec{C}_{\neq}]$, one case of A and C.	H) $[\{(e_i, e_j) : (q_k, ..., q_m)\}]$, one case of E and G.

Table 2.12: The eight cases and used to categorize datasets based on the number of categorical and quantitative fields, and accompanying data structures.

propositional formula. However, conditionals do not change the structure or interpretation of visualizations, and are the key to enumerating multiple visualizations.

2.7 Automation: Enumerating, Filtering and Scoring Visualizable Data Structures

Proceeding from the formalism just laid out in the previous sections, we can start approaching the central problem of presenting visualizations to the user. This has three steps: first, enumerating possible visualizations, then filtering visualizations, and finally scoring visualizations of different types and visualizations of the same type.

2.7.1 Enumerating possible visualizations

We start with the process of mapping from a dataset $\mathcal{D}$ to a set of visualizable data structures $\mathcal{V}$, following the procedures laid out in the previous section and summarized in Table 2.12. Then, we map from $\mathcal{V}$ onto visualization specifications $\mathcal{S}$ by using a visualization grammar, like in Table 2-4. Our specifications, then, are dictionaries containing visualization types, associated data structures, and functions used to arrive that those data structures.

2.7.2 Filtering visualizations

Visualizations are filtered out based mainly on visual interpretability. If the number of visual elements exceed a threshold that is renderable or useful (e.g. > 1000 nodes on a network visualization), we remove the specification. This threshold is determined through experimentation and the capabilities of browsers on commodity hardware.

2.7.3 Scoring visualizations

The last step, with this list of possible and interpretable visualizations, is to sort them in a way corresponding to their usefulness, interestingness, or meaningfulness. Our approach to this problem is to pre-compute descriptive or statistical properties of enumerated data structures. For instance, for simple 2-D scatterplot visualizations we can find the correlation of one axis against another. Or, we can test for normality of different histograms. However, is correlation, anti-correlation, or no correlation interesting? This depends on the dataset on the user, but at the least we can compute metrics that are proxies for interestingness and allow the user to search, sort, and filter on this metric.

The interestingness metrics we compute depend on dimension and type of visualization. For visualizations based on aggregations and result in a data structures with a single quantitative dimension, we can compare aggregations visualized on specific columns against aggregations against all other columns, inspired by the SeeDB system[17]. That is, we define a similarity function F_s that accepts two data structures V_i and V_j and returns the similarity between those two data structures. Further, we can compute the normality p-value, Gini coefficient of inequality, entropy, and model fits of the distribution,

If this aggregation results in a list of two dimensional quantitative tuples or a key-value pair of quantitative variables, we then have two quantitative vectors $\vec{Q}_1$ and $\vec{Q}_2$. With these two vectors, we compute the Pearson correlation $\text{corr}(\vec{Q}_1, \vec{Q}_2))$ and a linear regression of the two variables $\text{regress}(\vec{Q}_1, \vec{Q}_2))$ If the number of quantitative dimensions is greater than two, then perform a linear regression across all the variables, using the key (if available) as the dependent variable. The regression model used can change based on user input, computational feasibility, and regression results. For instance, if all regressions result in high p-values and little variance explained, we can continue try a different model.

2.8 Dealing with Multiple Datasets

2.8.1 Relationship Inference

Beyond capturing objects and attributes of objects, we are also concerned with relationships between objects. Here we are concerned with three types of relationships:

- **One-to-one** (one person is associated with one birth certificate)
- **One-to-many** (one person has one biological mother, but one mother may have many biological children)
- **Many-to-many** (one person many see several doctors, and one doctor may see several patients.)

The combination of the set of relationships $\mathcal{R}$ corresponding to datasets $\mathcal{D}$ and the properties of each dataset $\{\mathcal{P}_\mathcal{D}\}$ comprise the data model, or ontology. Obtaining an accurate and comprehensive data model the crucial first step in DIVE.

Relationships are crucial to visualizations because neglecting relationships between datasets can leave out a significant amount of possible visualizations. First, we're assuming that relationships exist only on entities, and thus only on categorical variables. Second, relationships are inferred taking the following heuristics into account:

1. Matching header names
2. Matching types
3. Similarity between uniqued data vectors.
4. Comparison of data vector cardinalities.

Given this model of relationships, we rely on un-pivoting long datasets and joins on multiple datasets so that, effectively, we are dealing with single long datasets.

Chapter 3

Implementation

A software tool incorporating the previous ideas and principles can be manifested in many forms on many devices. DIVE could plausibly be a desktop application, an extension for an existing desktop application, a mobile application, a browser plug-in, or a command-line tool. Instead of these forms, DIVE is being developed as a web application because of: time to develop, accessibility of development skills, testability, and internet penetration and the ubiquity of sufficiently powerful browsers.

DIVE is currently developed as a web application with clear distinctions between user interface, API, data access, data query, and data storage. DIVE is developed in accordance with the principles of modularity, safety, and incrementalism and designed according to the principles of discoverability, time-to-visualization, and shallow learning curves. This section serves two purposes: first to describe the state of DIVE as a platform, and second to generally describe an implementation of the previous ideas in a user-facing application.

3.1 Architecture

In general, our approach is inspired by the data state model described by Chi,[16]. We store all data descriptors, properties, visualization specifications, analysis specifications, and exported result specifications. A high-level architecture (not fully implemented) is shown in Figure 3-1. The data flows from top to bottom, with black horizontal separating conceptual stages. The first stage is data ingestion, in which data is read and stored. The second stage involves the detection of the data model / ontology. In other words, here we compute all the metadata necessary to enumerate visualization and analysis specifications In the third stage, we enumerate, score, and filter the specifications. Lastly, we expose these inferred specifications to the DIVE front-end. Here, a user can search through the specifications (right now, through the builder interface) and select specifications of interest.

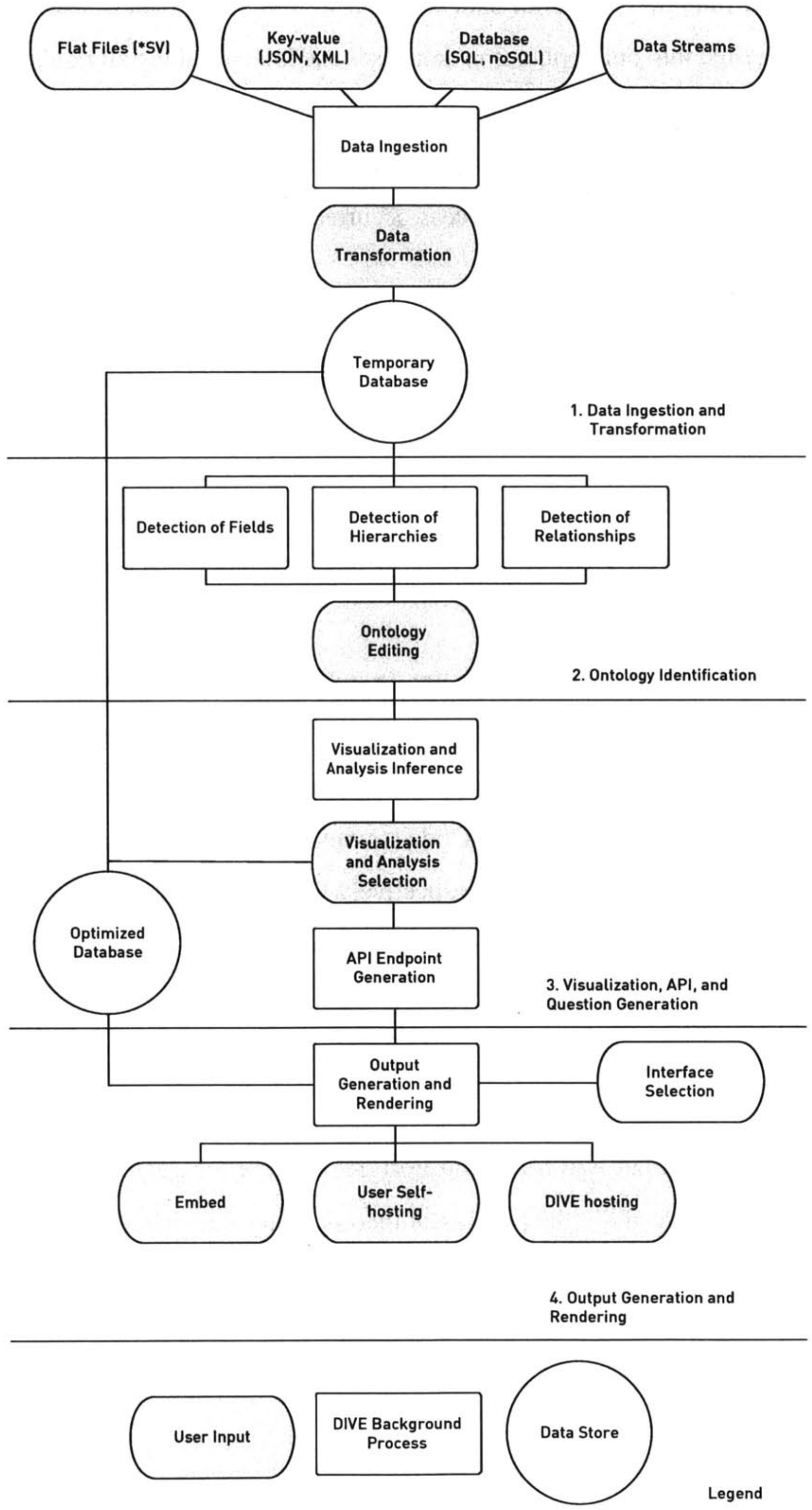


Figure 3-1: DIVE data flow and high-level architecture, read downwards from the top. Circular nodes denote data stores, white rectangles denote queued background processes and analyses, gray rounded rectangles denote stages at which output is presented to the user and during which the user can provide user input.

We prescribe to the *data lake* model of data storage,[1] in which user data is always stored in raw form before being ingested into our application databases and indexed along all fields if size permits. Data access in performed through the Python Blaze[2], and all datasets of reasonable size are read into memory as a Pandas data frame[3] or Numpy matrix[4]. By doing so, we can take advantage of accessible analysis libraries, minimize read/access occurrences, and manipulate data with a unified interface.

3.2 Technology Stack

In general, it is important for us to use modern web technologies. But because we we are still developing a first prototype, we choose components of our stack based on ease-of-use and speed of implementation.

The front-end of DIVE is implemented with the AngularJS javascript framework[5] in order to allow two-way data-binding, modular implementation, and clear code distinctions between interface, visualization and visualization. Interface elements use Angular Material which AngularJS components described by Google Material Design.[6]. Code is written in Coffeescript and LESS, which are augmented javascript and CSS preprocessors, respectively. Visualizations are developed using D3[7], D3Plus[8], and MetricsGraphics[9], depending on customization and visualization type.

The backend API is built using the python web framework Flask[10], wrapped in Gunicorn[11] or Nginx[12] depending on deployment environment. At the data access layer, we use the Python scientific computing libraries Pandas, Numpy, Scipy, Statsmodels, and Blaze. Finally, for persistence we are currently using MongoDB but will transition PostgreSQL. For messaging and queueing background tasks in this system, we are in the process of incorporating RabbitMQ[13].

[1]http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/data-lakes.jhtml
[2]http://blaze.pydata.org/en/latest/
[3]http://pandas.pydata.org/
[4]http://www.numpy.org/
[5]https://angularjs.org/
[6]https://material.angularjs.org,https://www.google.com/design/spec/material-design
[7]http:d3js.org
[8]http://d3plus.org
[9]http://metricsgraphicsjs.org/
[10]http://flask.pocoo.org
[11]http://gunicorn.org/
[12]http:://nginx.org
[13]http://www.rabbitmq.com

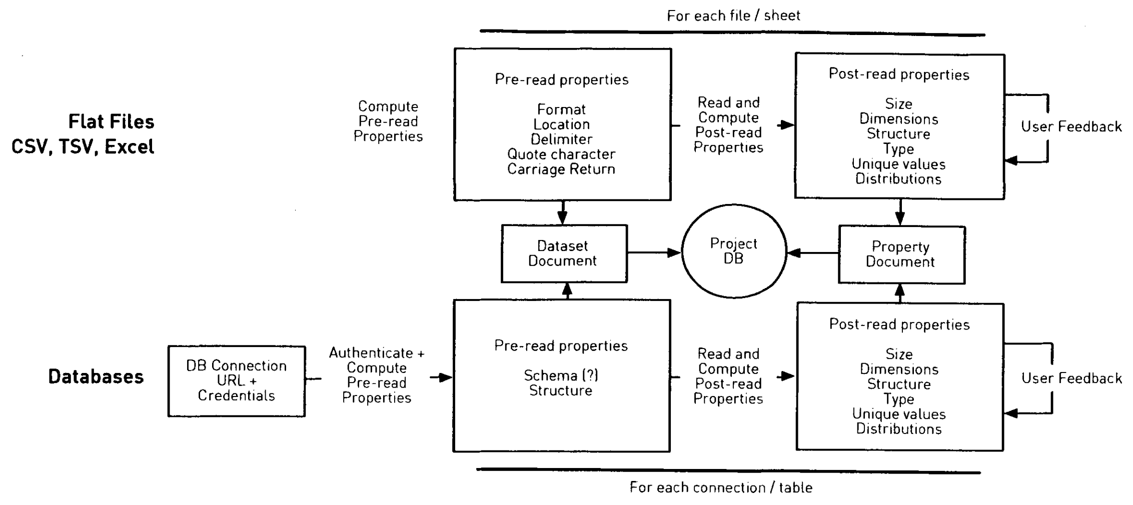


Figure 3-2: Data ingestion and property inference stages.

3.3 Data Pipeline

A user interacts directly with a dataset in four ways: initial upload, access (read), modification, and deletion. Between a user uploading a dataset and DIVE storing the dataset in a persistent form, its necessary to compute and record properties of the dataset necessary to read it, as well as the properties of the dataset after reading it. This data ingestion scheme is shown in Figure 3-2.

At this stage, we create pre-read and post-read property documents for each dataset. Upon a new data upload or on user feedback, an ontology document is generated that captures the current state of the data model, and then associated with a user's project.

3.3.1 Data Ingestion and Property Inference

Data ingestion in DIVE includes four sequential processes: first, a user uploads a dataset or a locator to database, DIVE storing this dataset; second, DIVE read this data or a sample of this data into memory; third DIVE analyses this data; and lastly, DIVE then returns a sample of this data. The analysis is the most important part for later visualization and analysis, and is shown in Figure 3-2. Proceeding from either a flat file or a database, we first analyze properties of datasets that we need to read the dataset, or that we can get without reading the dataset. For flat files, this would include the file type, size, and delimiters. For databases, this includes the schema (if it exists). After reading the respective data, then we compute the types and distribution of each column in the data. Small datasets are read into memory at this step. Large datasets are analyzed by sampling both randomly and by the first N lines, the results of which are then compared.

3.4 User-facing Design

As any user-facing application, design is a key component of DIVE. Design is especially important in an application trying to implement ideas of intelligence alongside principles of transparency, two ideas that can at times be counter to each other. By design, we mean the design of interfaces (how components look), experiences (how workflows feel), and content (what is shown, and when).

3.4.1 Interface Design

The interface of DIVE is based on the dashboard paradigm seen in most data manipulation tools. The general layout of DIVE is shown infigs. 3-4 to 3-8. Within a specific project, the top-most blue bar is used to navigate between viewing data, visualizations, and analyses. At each stage, the secondary top-bar is used to manipulate and filter elements on the page, or to navigate between secondary pages.

3.4.2 Experience Design

There are two user "modes" in DIVE, the top-level mode and the project-specific mode. In the top-level mode, users can create new projects, view currently available projects, and browse public examples from other projects. We are optimizing the workflow of DIVE to minimize the time for a user to arrive at a visualization and analysis. In practice, this involves including features such as predefined datasets and automatically creating projects for first-time users. After a user has navigated to a project, or has created a new one, the first pane is the interface for uploading and interacting with specific datasets. After a user has uploaded datasets, he can arrive at the un-filtered galley view shown in Figure 3-4 in order to filter down visualizations by a pre-defined grammar.

Given the importance of user experience to the success of user-facing applications, and the specific necessity of user input in our framework of filtering down enumerated visualizations, we have described specific grammars for the filtering of visualizations and analyses. An example of the workflow used in building visualizations is shown in Figure 3-3, which shows the steps necessary for filtering down enumerated visualizations.

3.4.3 Content Design

The visual content of DIVE is centered on the visualization and the data used to create this visualization, as seen in the single-visualization pages in figs. 3-7 and 3-8. Here, we are optimizing the amount of the screen dedicated to the specific visualization, while allowing easy access to the data used to generate the visualization, formatted as a table.

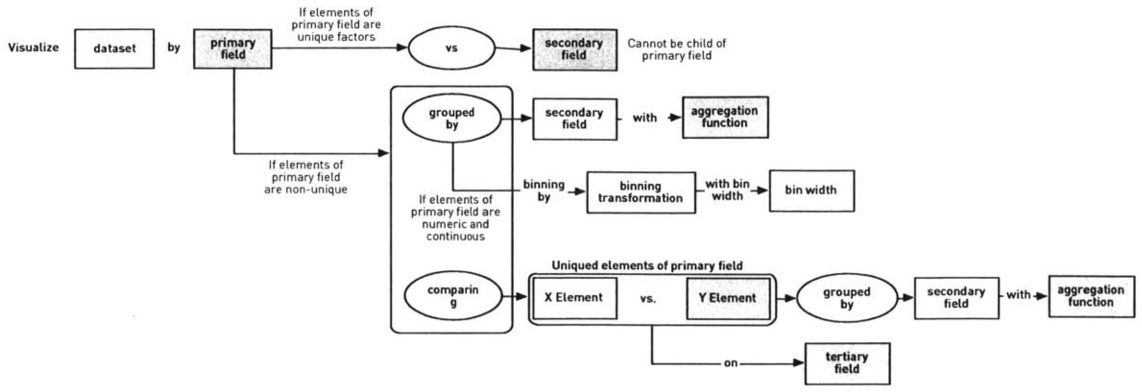


Figure 3-3: Decision tree and user interaction scheme in the visualization builder stage. Gray boxes indicate free parameters that a user does not need to specify. If not specified, DIVE will return the visualization data according to all possible visualizations in accordance with the previous clauses.

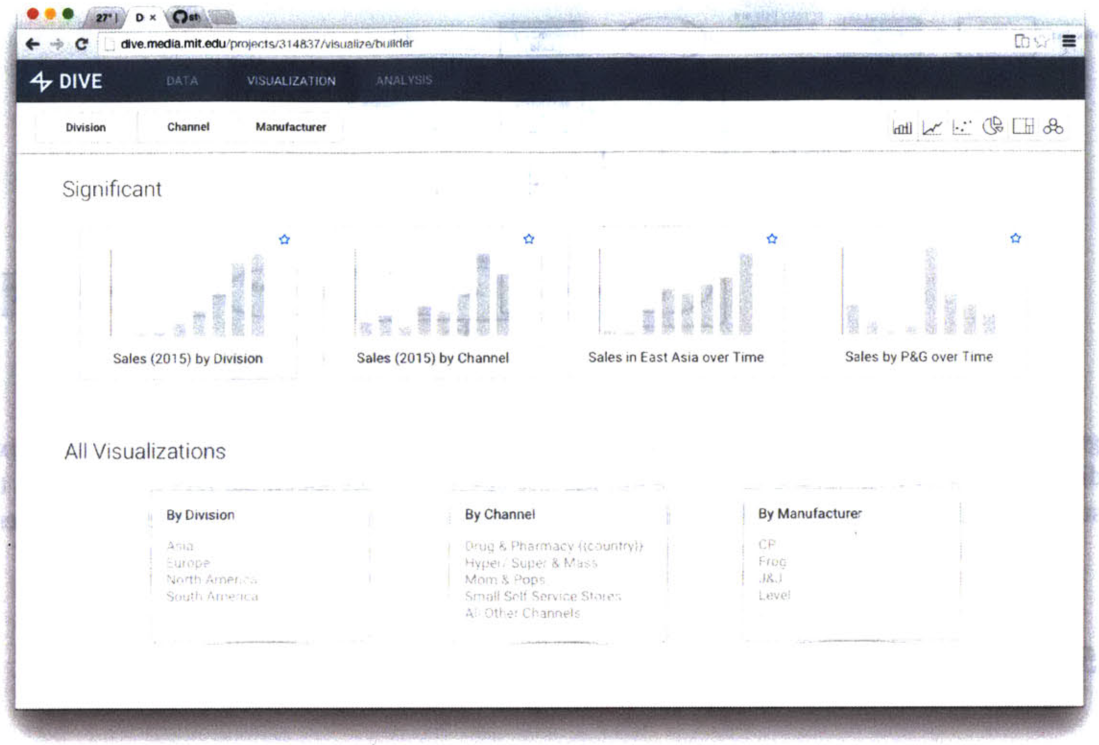
3.5 Future Work

DIVE is currently in preparation for an alpha MVP by end of summer 2015, and release as a publicly available as a beta platform by the end of 2015. For the alpha MVP, we are aiming to support more data sources (e.g. network data formats, APIs), more visualization types, and a robust user-feedback loop for users interacting with inferred data properties. By the beta platform, we will implement queueing and messaging features for asynchronous data processing, robust session management, encryption, authentication, public datasets, and analysis and visualization export capabilities. Future work beyond 2015 depends on feedback from the 2015 beta.

In parallel, we will be continuing to collaborate with Colgate-Palmolive company to ensure that DIVE encompasses their use cases and that our visualizations and statistical analyses have business impact. In particular, this may involve developing individual, internal deployments of DIVE.

With respect to the conceptual work of DIVE, depending again on the 2015 beta, we will be drafting a paper documenting the formalism and releasing open-source libraries implementing specific aspects of the data ingestion and automatic enumeration of statistical analyses and visualizations. These packages will be modular versions of the backend functionality that already exists within DIVE.

A table showing the features that currently exist, and those that we plan for the MVP and final version are shown in Table 3.1.



2a. Visualizations — Gallery

After uploading datasets, you're taken to the Gallery. The Gallery lists significant/recommended visualizations at the top. All other visualizations are grouped by Entity below. Clicking on an entity group is the same as filtering on it in the filter bar above.

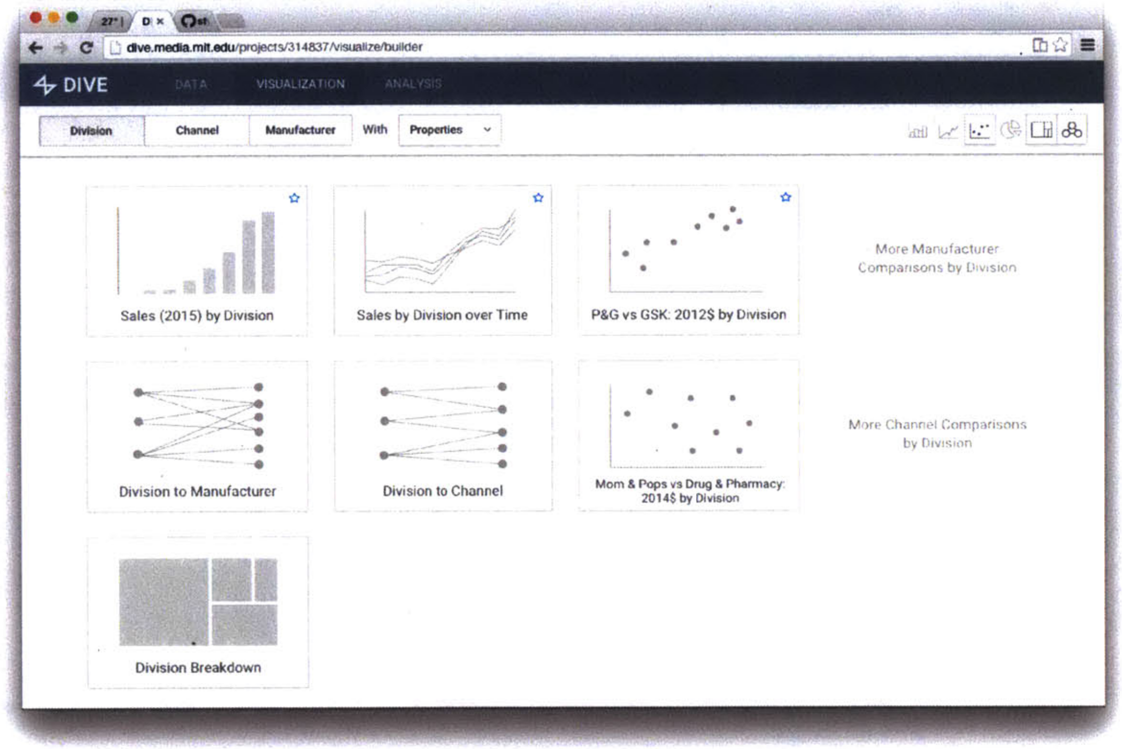
Figure 3-4: Unfiltered multi-visualization gallery view.

3.6 Discussion

Here we have documented the development of DIVE, a web-based software platform, planned for public release, that implements the automatic visualization functionality detailed in the previous section. DIVE is designed to minimize the time necessary for a user to see a result, and to maximize the amount of results a user can digest on a single page. DIVE is implemented modularly so that, eventually, other applications could be built on top of DIVE's exposed API endpoints.

The main implementation concerns we face in developing DIVE are the same as those facing any other "big data" tool. In this case, we can address each the three types of big data: high variance in type/structure, high velocity, and high volume.

DIVE accommodates many types and structures of data, though it is difficult to work with *every* dataset that exists. This is mainly a development issue, and though it is our hope that we can develop more data adapters (e.g. for Salesforce, SAP, and different databases), our first concern is testing the usability of our workflow and design at scale.

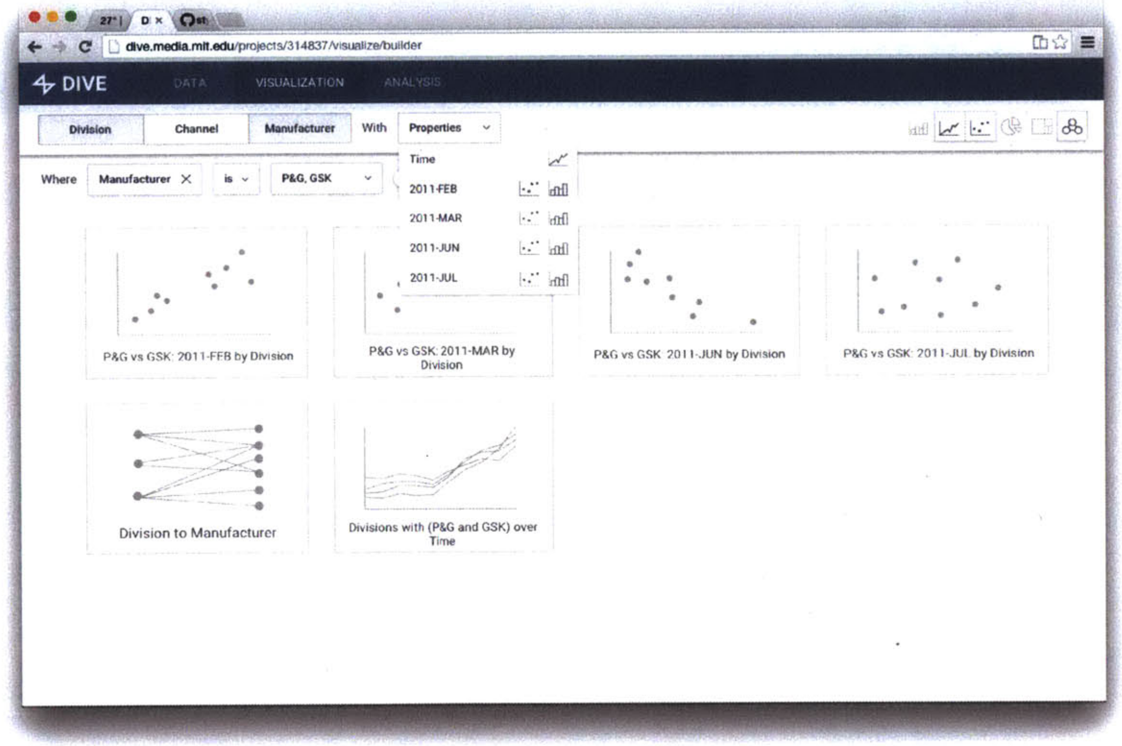

dive.media.mit.edu/projects/314837/visualize/builder
DIVE
DATA
VISUALIZATION
ANALYSIS
Division
Channel
Manufacturer
With
Properties
Sales (2015) by Division
Sales by Division over Time
P&G vs GSK: 2012$ by Division
More Manufacturer Comparisons by Division
Division to Manufacturer
Division to Channel
Mom & Pops vs Drug & Pharmacy: 2014$ by Division
More Channel Comparisons by Division
Division Breakdown

2b. Visualizations — Gallery, select first entity

The user selected the "By Division" group, which filtered on the available visualizations. Some (but not all) individual comparisons between two entities are shown, with a link to see more comparisons.

h

Figure 3-5: Multi-visualization gallery view without selecting secondary field.

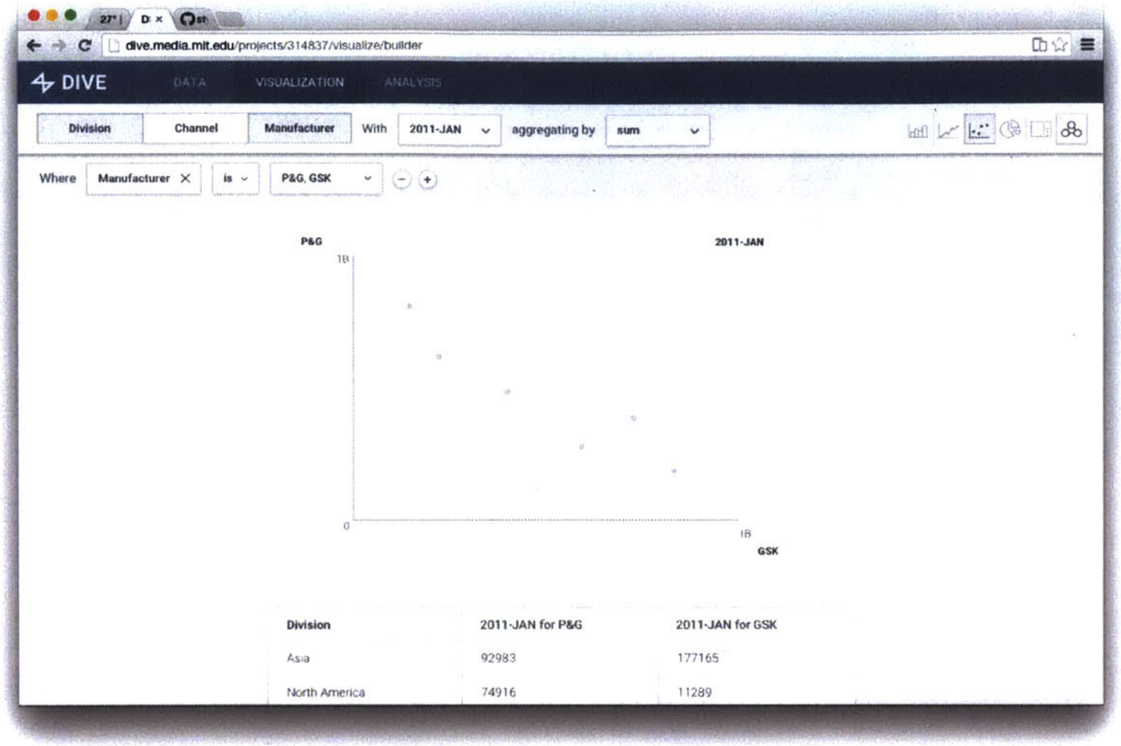

dive.media.mit.edu/projects/314837/visualize/builder
DIVE
DATA
VISUALIZATION
ANALYSIS
Division
Channel
Manufacturer
With
Properties
Time
2011-FEB
2011-MAR
2011-JUN
2011-JUL
Where
Manufacturer
is
P&G, GSK
P&G vs GSK: 2011-FEB by Division
P&G vs GSK: 2011-MAR by Division
P&G vs GSK: 2011-JUN by Division
P&G vs GSK: 2011-JUL by Division
Division to Manufacturer
Divisions with (P&G and GSK) over Time

2c. Visualizations — Gallery, select second entity

The user selected the "More Manufacturer Comparisons by Division" group, which filtered on the available visualizations. It auto-selected a particular conditional on Manufacturers being P&G and GSK, but the user can change this.

h

Figure 3-6: Gallery view before selecting secondary field.



Division	2011-JAN for P&G	2011-JAN for GSK
Asia	92983	177165
North America	74916	11289

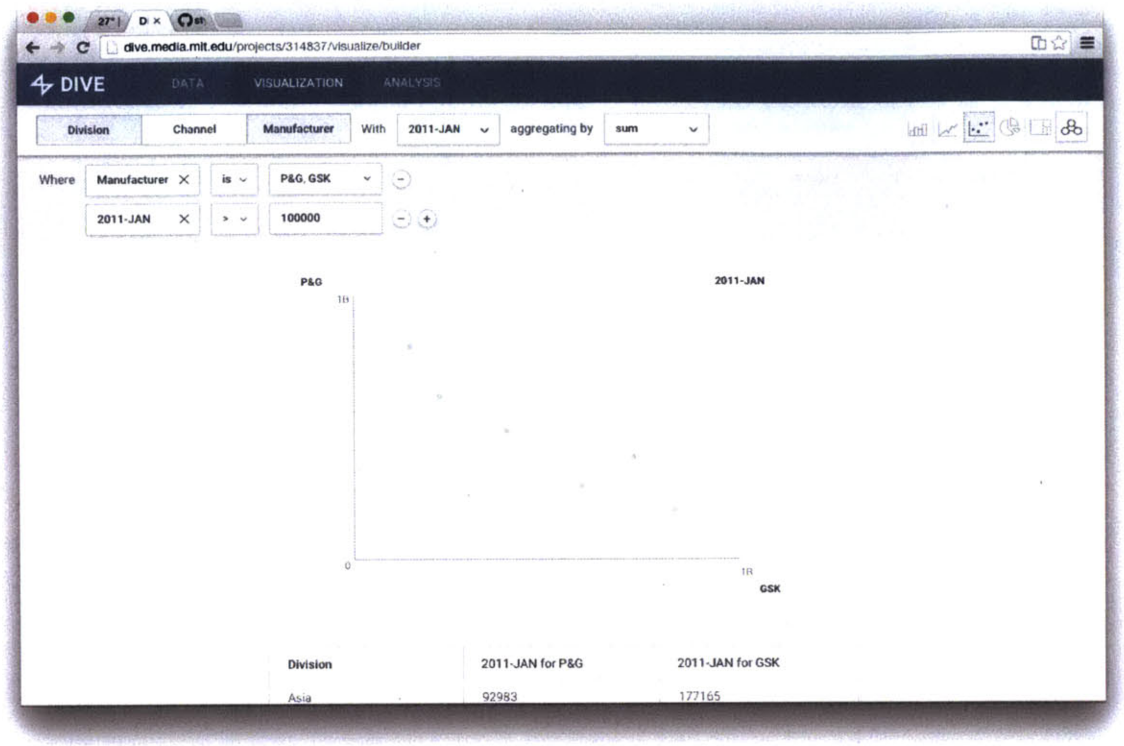
2d. Visualizations — View visualization

The user selected a particular visualization. If applicable, they can choose how they want to aggregate data in the graph. A datatable shows the individual values of the points below.

Figure 3-7: Single-visualization view without conditionals.

While DIVE does not address real-time data in its current state, its data pipeline will be able to work with real-time data given some degree of persistence of its data model and a "pool" of streamed data. We will accommodate real-time data using active polling on the back-end and an updating data model based on pooled data. This solution depends on the speed and volume of incoming data, but from early experiments it returns meaningful results for real-time APIs.

Lastly, while each stage in DIVE takes a reasonable amount of time (up to 10 seconds) for datasets up to 100MB ($\sim$100,000 rows) on commodity machines, some of the operations necessary for constructing data models and enumerating visualizations scale non-linearly with the number of fields in a dataset. Further, more expensive statistical operations scale non-linearly with the number of elements in its arguments. Because of this limitation, given a user's datasets, it is necessary to profile the amount of time each process takes, and perform some processes asynchronously.



2e. Visualizations — View visualization, add conditional

Conditionals can be used to filter the data shown in a visualization. The visualization-type toggle buttons can be used to switch between different types of visualizations.

Figure 3-8: Single-visualization view, after filtering from the gallery and specifying conditionals.

	Front-end Features	Back-end Features
Current	Project creation, DIVE-specific authentication, data uploader and previewer, visualization builder, partial analysis builder	Type detection, hierarchy detection, ontology detection, specification enumeration, specification scoring, visualization data creation from specification C, regression calculation
MVP	Visualization export, comprehensive visualization integration, analysis export, more analysis type support, public datasets	Compatibility with all flat file data types, messaging system, queueing system, asynchronous tasks, task profiling
Final	OAuth authentication, public/private project separation, public project viewing, walkthrough, function tooltips, specification-to-natural language conversion, full interactive website assemble and export, multiple export types, manual statistical model into	Data cleaning features, integration with other databases, real time APIs compatibility, learning from user visualization and analysis selection (personalization), support for data changes

Table 3.1: The eight cases used to categorize datasets based on the number of categorical and quantitative fields.

Chapter 4

Evaluation

A DIVE beta test to the larger visualization community is planned for September 2015. But because DIVE is not yet publicly available, evaluation on the current state of the platform relies on versatility to different datasets, speed to arrive to specific visualizations or analyses, and qualitative feedback.

4.1 Versatility

DIVE is developed in collaboration with the Colgate-Palmolive company, a multinational consumer products company focused on home and oral care, who have generously provided five research and development datasets related to regional toothbrush sales, bacterial genomics, patent research, product stability and clinical oral research. Of these five datasets, the former three are based on single databases, while the latter two are composed of multiple datasets. Further, the datasets vary in structure. The sales dataset a wide time series dataset with 58,000 rows. The genomic dataset is a long data set with 500 rows and a single time column. The patent research dataset is an extremely dense, long dataset with 2,500 rows. The product stability dataset consists of thirty-three files each containing with a long time series dataset of stability metrics per batch of product. Lastly, the clinical research dataset is four separate studies with hundreds of rows. Using DIVE, we have successfully ingested and visualized all of these datasets.

4.2 Qualitative Evaluation of Usability and Speed

The two primary metrics we're concerned with are usability and speed. That is, given a task, is a user able to achieve that task without becoming confused or blocked, and if so, did it take a reasonable amount of time? Further, given that DIVE implements a user workflow that may be foreign to users familiar with other visualization tools, what is the threshold for gaining a working understanding the different abstractions?[27]

The current version of DIVE was evaluated by three graduate students and one software engineer working in industry. Participants had varying levels of visualization expertise and background. First, these five individuals were given a computer running DIVE locally, a synthetic dataset (long-format, two numeric columns and one categorical column), and a brief overview of the motivation behind DIVE without a tutorial about the various workflow stages. Then, they were told to complete five sequential tasks: 1) upload the dataset, 2) check inferred types and data model, 3) generate a treemap visualization, and 4) run a simple linear regression. Participants were given twenty minutes to complete the tasks and were encouraged to "think-aloud" while completing the tasks.

4.2.1 Results

All five participants completed the five assigned tasks, with varying levels of difficulty. Some difficulties were expected, others unexpected, though all were instructive in informing our future design and evaluation. The most common difficulty was for users to understand the language used in the "visualization" page, such as the difference between "comparison" and "vs." after selecting a dataset to visualize and a primary field in visualization. One participant, even after understanding the distinction, said "In my mind, comparison is always vs.". This confusion was expected, and though the confusion was resolved for two of five participants by simply trying out both options, in the future we will create more descriptive language and a walk-through feature and in-line explanations.

Another pain-point was the transition between data upload and visualization or analysis. That is illustrated by a reaction of one participant after uploading a dataset and observing inferred types, who asked "what exactly do I do now?" This difficulty was surprising given our design of the top bar, and will be addressed by either implementing a "notification" feature that highlights the visualization or analysis steps with the number of results shown in the respective galleries.

Lastly, two participants were confused about the relationship between "Visualization" and "Analysis" in transitioning between the third task of creating a treemap visualization and the fourth task of running a simple linear regression. To summarize one participant's comments, it was confusing to navigate between the two because he expected visualization and analysis to both be available in a single mode, like in most data applications. This feedback is not as easily resolvable as the previous two, though it may be addressed in the future by closer integration of the two modes (e.g. being able to navigate between scatterplots and corresponding regressions).

On the whole participant reaction to DIVE was encouraging, even from those who had difficulty. Many participants said that the idea of enumerating visualizations and analyses "made a lot of sense" and that it was surprising that "this doesn't already exist, at least as...a MATLAB application." Even though this evaluation had a small number of participants, participant feedback has been extremely helpful in revealing the language and workflow that DIVE's developers have taken for granted, but that may confuse and block users. By the end of Summer 2015 MVP version, we plan to conduct more thorough, quantitative evaluation.

Chapter 5

Conclusion

In this thesis we have introduced DIVE, a framework and web-based platform for automatically creating visualizations and analyses of arbitrary structured datasets. DIVE implements a new framework for exploratory data visualization and analysis as a series of data ingestion and processing modules, supporting the creation of the most commonly used 2D web visualizations, such as treemaps, scatterplots, barcharts, and time series. In particular, DIVE contributes scalable abstractions and API endpoints for data I/O, retrieving specifications for possible visualizations and analyses given a dataset, and returning results corresponding to specifications.

DIVE is part of a larger move to rethink the outcome-centric focus in current data tools and workflows and bring more automation into data analysis and exploration. In future work, we plan to introduce more powerful feedback loops for users to iterate on inferred data structures, provide more data adapters, implement a new analytic data engine for query optimization, allow for visualization export and sharing, and support more visualization types.

Bibliography

[1] Wikipedia. Moore's law — wikipedia, the free encyclopedia, 2015. [Online; accessed 21-July-2015].

[2] Bruno Latour. Visualisation and Cognition: Drawing Things Together . *Knowledge and Society Studies in the Sociology of Culture Past and Present*, pages 1–33, November 2011.

[3] EMC. Emc digital universe study with research and analysis by idc.

[4] S Kandel, A Paepcke, and J M Hellerstein. Enterprise data analysis and visualization: An interview study. *Visualization and ...*, 2012.

[5] B A Price, R M Baecker, and I S Small. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, 1993.

[6] C Daassi, L Nigay, and M C Fauvet. A taxonomy of temporal data visualization techniques. *Information-Interaction-Intelligence*, 2005.

[7] Edward R Tufte. The Visual Display of Quantitative Information, 1992.

[8] Stuart Card, Mackinlay Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization.* Using Vision to Think. Turtleback, January 1999.

[9] Min Chen, David Ebert, Hans Hagen, Robert S Laramee, Robert Van Liere, Kwan-Liu Ma, William Ribarsky, Gerik Scheuermann, and Deborah Silver. Data, Information, and Knowledge in Visualization. *Computer Graphics and Applications, IEEE*, 29(1):12–19, 2009.

[10] Jean-Daniel Fekete. The InfoVis Toolkit. *IEEE Symposium on Information Visualization*, pages 167–174, 2004.

[11] Hadley Wickham, Dianne Cook, Heike Hofmann, and Andreas Buja. Graphical inference for Infovis. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):973–979, November 2010.

[12] Stuart K Card, George G Robertson, and Jock D Mackinlay. *The information visualizer, an information workspace*. ACM, New York, New York, USA, April 1991.

[13] Jeffrey Heer, Stuart K Card, and James A Landay. *prefuse: a toolkit for interactive information visualization*. a toolkit for interactive information visualization. ACM, New York, New York, USA, April 2005.

[14] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[15] M Livny, R Ramakrishnan, K Beyer, G Chen, D Donjerkovic, S Lawande, J Myllymaki, K Wenger, D Donjerkovic, S Lawande, J Myllymaki, and K Wenger. *DEVise: integrated querying and visual exploration of large datasets*, volume 26 of *integrated querying and visual exploration of large datasets*. ACM, June 1997.

[16] Ed H Chi. A taxonomy of visualization techniques using the data state reference model. *IEEE Visualization 2000*, pages 69–75, 2000.

[17] Manasi Vartak, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. SeeDB: automatically generating query visualizations. *Proceedings of the VLDB Endowment*, 7(13):1581–1584, August 2014.

[18] A Parameswaran and N Polyzotis. Seedb: Visualizing database queries efficiently. In *Proceedings of the VLDB . . .*, 2013.

[19] J D Mackinlay and P Hanrahan. Show me: Automatic presentation for visual analysis. . . . *and Computer Graphics*, 2007.

[20] Pat Hanrahan. *VizQL: a language for query, analysis and visualization*. a language for query, analysis and visualization. ACM, New York, New York, USA, June 2006.

[21] P Hanrahan, C Stolte, and J Mackinlay. visual analysis for everyone. *Tableau White paper*, 2007.

[22] Richard Wesley, Matthew Eldridge, and Pawel T Terlecki. An analytic data engine for visualization in tableau. *the 2011 international conference*, pages 1185–1194, June 2011.

[23] Eugene Wu, Leilani Battle, and Samuel R Madden. The case for data visualization management systems: vision paper. *Proceedings of the VLDB Endowment*, 7(10):903–906, June 2014.

[24] Jeffrey Heer and Michael Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *ACM Human Factors in Computing Systems (CHI)*, pages 203–212, 2010.

[25] Jacques Bertin and William J Berg. *Semiology of Graphics*. Diagrams, Networks, Maps. Esri Press, 2011.

[26] William Kent. *Data and Reality: Basic Assumptions in Data Processing Reconsidered*. Elsevier Science Inc., New York, NY, USA, 1978.

[27] B Myers, S Hudson, and R Pausch. Past, present and future of user interface software tools. 1999.