Augmenting Exploratory Data Analysis with Visualization Recommendation

Kanit Wongsuphasawat

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Jeffrey Heer, Chair

Bill Howe

Jock Mackinlay

Program Authorized to Offer Degree:

Computer Science & Engineering

University of Washington

Abstract

Augmenting Exploratory Data Analysis
with Visualization Recommendation

Kanit Wongsuphasawat

Chair of the Supervisory Committee:

Professor Jeffrey Heer

Paul G. Allen School of Computer Science & Engineering

Exploratory data analysis is one of the key activities for understanding and discovering new insights from data. As exploratory data analysis can involve both open-ended exploration and focused question answering, analysis tool should facilitate both exploration breadth and analysis depth. However, existing data exploration tools typically require manual chart specification, which can be tedious and prevent analysts from rapidly exploring different aspects of the data. Moreover, analysts may be blindsided by their own cognitive biases and prematurely fixate on specific questions or hypotheses. Without discipline and time, analysts may overlook important insights in the data, such as potentially confounding factors and data quality issues, and produce inaccurate results in their analyses.

To help analyst perform rapid and systematic data exploration, this dissertation presents the design of mixed-initiative systems that complement manual chart specification with chart recommendation.

To better understand the practice and challenges of exploratory data analysis, we first conduct an interview study with 18 data analysts. From the interview data, we characterize the goals, process, and challenges of exploratory data analysis. We then identify design opportunities for exploratory analysis tools. One major opportunity is facilitating rapid and systematic exploration with automation and guidance. The rest of the dissertation addresses this opportunity by contributing a stack of systems to augment exploratory analysis tools with chart recommendation.

At the foundations of this stack, we introduce new formal languages for chart specification and recommendation. The *Vega-Lite* visualization grammar provides a representation for specifying and reasoning about charts. Building on Vega-Lite, the *CompassQL* query language combines partial chart specification with recommendation directives to provide a generalizable framework for chart recommendation via queries over the space of visualizations.

Based on these foundations, we used the iterative design process to develop and study new recommendation-powered visual data exploration tools. *Voyager* enables data exploration via browsing of recommended charts, while allowing users to steer the recommendations by selecting data fields and transformations. Our user study, which compares Voyager with a traditional chart authoring tool, indicates the complementary benefits of manual authoring and recommendation browsing. Inspired by the study result, *Voyager 2* blends manual and automated chart authoring in a single tool to facilitate rapid and systematic data exploration while preserving users' flexibility to directly author a broad range of charts.

All of these systems have been released as open-source projects and adopted by both research and professional data science communities.

Table of Contents

List of Figures

Acknowledgments

First of all, I would like to thank my family, especially my parents, for their unconditional support since I was born. My PhD journey here in Seattle also led me to meet my fiancée, Dear, who has since provided me daily support, joy, and love. I definitely have to thank my elder brother, Krist. Without him, I might not even be writing this thesis. Long story short, I was studying for my master's degree at Stanford while Krist was a PhD student at the University of Maryland, with Ben Shneiderman. Though I grew up having Krist as both my best buddy and a very kind elder brother, I felt like it must be a bad idea to do PhD in the same research field as his. However, Krist insisted that I should at least take the Data Visualization course at Stanford because the professor there was the best researcher in the field.

As you might guess, the professor Krist talked about was Jeffrey Heer, my PhD advisor. His Data Visualization class was one of the most fun classes I have ever taken. More importantly, it convinced me that I could impact and change people's lives through research, basically changing the way people work with data in our case. As I knew that Jeff was moving to the University of Washington (UW), I applied to UW and became his first UW PhD student. Fast forward to 5 years later, I think I have achieved what I hoped to do. Our work, especially Vega-Lite, has been widely adopted both in research and in industry. Voyager has also become a part of Jeff's picture on Wikipedia! (Please do not troll and change his picture!) Over the

years, I have learned from Jeff to be a better researcher, engineer, designer, and a better person. Despite his duty as a professor and co-founder of his startup, Jeff has led by example to show what it takes to make an impact through research by developing Vega, which is the foundation of the Vega-Lite, CompassQL, and Voyager systems in this thesis. It is hard to cover all the great things about Jeff, but I can definitely say that I am so fortunate to have an advisor who is both smart and caring.

Over the years, I have had the luxury of support from many mentors. I would like to give special thanks to Jock Mackinlay, Bill Howe, and Anushka Anand for their mentorship throughout these 5 years, especially to Jock who generously let us build on the idea from my internship at Tableau. I also thank other members of Tableau Research, members of Trifacta, as well as Martin Wattenberg, Fernanda Viégas and the Big Picture Group at Google for invaluable internship experience.

Many say that a PhD life can be lonely as PhD students often have to work alone for 5 or more years, but that has not been the case for me. For all of these projects, I have been lucky to collaborate with many amazing colleagues, especially with Dominik Moritz who co-authors all systems in my thesis. I am also grateful to have worked closely with Arvind Satyanarayan, Yang Liu, Younghoon Kim, and Zening Qu. Moreover, I appreciate the feedback and encouragement from the rest of the Interactive Data Lab and the HCI group members. Throughout the years, I have had many wonderful collaborators in the open source community including Jake VanderPlas, Brian Granger, Ji Zhang, Saul Shanabrook, Jim Vallandingham, Yannick Assogba, K. Adam White, and Irene Ros. I am also thankful to have worked with and mentored many UW undergraduate research assistants and Google Summer of Code students including Will Strimling, Matthew Chun, Yuhan “Zoe” Lu, Youying Lin, Ayush Saraf, Lingyue “Cynthia” Zhang, Swojit Mohapatra, Akshat Shrivastava, Chanwut Kittivorawong,

Sira Horradarn, Melissa Diamond, Saba Noorassa, Souvik Sen, Felix Ouk, Shaheen Sharifian, Alan Banh, Fion Chan, and Halden Lin.

Finally, I would like to thank all my UW and Seattle friends for all the fun and friendship we have had over the years. In particular, I owe my thanks to the Thai community in Seattle—especially Aek, P'Toey, P'Luck, P'Pat, P'Kobe, P'A, Knot, Ton, Jump, June, and Amm—for making me feel like home here in Seattle.

Thank you!

1 Introduction

As an amount of data is growing exponentially [92], we are in an era of data abundance. One of the biggest challenge in this era is to understand and extract value from these data, as Hal Varian once said:

> "Now we really do have essentially free and ubiquitous data. So the [complementary] scarce factor is the ability to understand that data and extract value from it."
> - Hal Varian [4]

One of the key activities in data science to handle this challenge is *exploratory data analysis* (EDA) [135], a practice that primarily utilizes data visualization to understand data and gain new insights [29, 97]. In exploratory data analysis, analysts typically have to perform two high-level analysis strategies [64, 96]. First, they should begin with a broad, *open-ended exploration*, familiarizing themselves with the shape and structure of the data as well as checking potential data quality issues. After familiarizing with the data, they may then focus on more targeted *question answering*. Investigating these questions may spark further exploration of potentially relevant factors and outcomes, in turn leading again to more focused analysis. An ideal analysis should cover both exploration breadth and analysis depth.

However, in practice analysts may fail to achieve systematic coverage during exploration for many reasons. First, existing data exploration tools typically require *manual chart specification*, involving many tedious and non-trivial steps including choosing data fields, applying data

transformation, and designing visual encodings. As a result, this tedious process often prevents analysts from rapidly exploring different aspects of the data. Moreover, cognitive biases [78], such as confirmation biases [100], may lead analysts to prematurely fixate on specific questions or hypotheses. Finally, analysts often have limited time for data exploration due to external factors such as the pressure to complete their projects. For these reasons, analysts may overlook important insights in their data, such as potentially confounding factors and data quality issues, and produce inaccurate results in their analysis.

To facilitate rapid and systematic data exploration, visualization tools might automatically recommend a diverse set of charts for users to browse. However, there are many challenges for chart recommendation. For any given data table the choices of variables, transformations and visual encodings lead to a combinatorial explosion. Thus, appropriate filtering and recommendation strategies are needed to prune the space and promote relevant views. Moreover, automation is unlikely to succeed on its own. As exploration proceeds, analysts would apply their domain knowledge to interpret the structure of the data, and thus have specific interests on certain aspects of the data, requiring an interface that enables interactive steering of recommendations. To balance between automation and user control, intelligent visualization tools should adopt a *mixed-initiative* approach [71], allowing the users and the automated system to collaborate to achieve the users' goals.

1.1 Contributions

This thesis investigates the design of mixed-initiative exploratory analysis tools that complement manual chart authoring with recommendation to facilitate rapid and systematic exploration of tabular data. In particular, this thesis makes contributions in three categories:

1. **Characterization of goals, process, and challenges in exploratory data analysis.**
   To better understand the practice of exploratory data analysis, we conducted semi-

structured interviews with 18 data analysts. We characterize common exploration goals: *profiling* (assessing data quality) and *discovery* (gaining new insights). Though the EDA literature primarily emphasizes discovery, we observe that our participants mostly focus on discovery in the context of open-ended analyses, whereas all participants engage in profiling across all of their analyses.

We also describe the process and challenges of exploratory analysis highlighted by our interviews. We find that analysts must perform repetitive tasks (*e.g.*, examine numerous variables), yet they may have limited time or lack domain knowledge to explore data. Analysts also often have to consult other stakeholders and oscillate between exploration and other tasks, such as acquiring and wrangling additional data.

Based on the interviews, we identify design opportunities for exploratory analysis tools. One major opportunity is facilitating rapid and systematic exploration with automation and guidance. The rest of this thesis addresses this opportunity by introducing a stack of systems for augmenting exploratory data analysis tools with chart recommendation.

2. **The design of formal languages for chart specification and recommendation.**

   (a) We introduce the *Vega-Lite* visualization grammar as a formal model for specifying and reasoning about charts. Vega-Lite offers primitive building blocks for composing a broad range of visualizations. To provide concision, Vega-Lite allows specifications to omit low-level details and automatically infers default values for these omitted details. With a concise JSON syntax, Vega-Lite supports both rapid manual chart authoring and programming generation. Beyond supporting other systems in this thesis, Vega-Lite has also served as a platform for developing new applications and research projects.

   (b) We present the *CompassQL* query language to provide a general-purpose framework for chart recommendation via queries over the space of visualizations. To

describe a query, CompassQL combines a partial Vega-Lite chart specification with methods for grouping and ranking candidate charts. Given a query, the CompassQL query engine then enumerates, ranks, and groups candidate charts to produce an organized collection of recommended charts. We demonstrate the expressivity of CompassQL with examples queries that describe a variety of existing chart recommendation approaches.

3. **The design and study of mixed-initiative visual data exploration tools that are powered by chart recommendation.** With the new formal languages as foundations, we use the iterative design process to develop and study new mixed initiative systems that enable recommendation-powered visual data exploration.

   (a) We introduce *Voyager*, a mixed-initiative system that facilitates breadth-oriented data exploration with interactive navigation of recommended charts. Voyager exchanges specification for browsing, providing an organized display of recommended visualizations and enabling user input for both chart refinement and recommendation steering. To evaluate Voyager, we present a user study that compares Voyager with a standard chart authoring tool. The study result indicates the complementary benefits of manual authoring and recommendation browsing.

   (b) Inspired by the Voyager's user study, we introduce *Voyager 2*, an interactive system that blends manual and automated chart authoring in a single tool and enables users to alternate between open-ended exploration and focused question answering. Voyager 2 augments a standard chart authoring tool with two new partial specification interfaces: *wildcards* let users specify multiple charts in parallel, while *related views* suggest visualizations relevant to the currently specified chart. In a user study, we find that Voyager 2 leads to increased data coverage compared to a traditional specification tool, while still allowing analysts to flexibly drill-down and answer specific questions.

1.2 Outline

We begin in Chapter 2 by providing background on exploratory data analysis and discussing the results of our interview study on goal, process, and challenges of exploratory data analysis.

The remainder of this thesis presents systems and methods for augmenting exploratory data analysis with visualization recommendation.

Chapter 3 surveys prior work on exploratory search systems and tools for visualization specification and recommendation.

Chapter 4 presents the Vega-Lite visualization grammar, which serves as a representation for specifying and reasoning about charts in our recommender systems and user interfaces.

Chapter 5 introduces the design of the Voyager visualization browser and the Compass chart recommender system underlying the Voyager interface. This chapter also discusses the user study comparing Voyager with the PoleStar chart specification tool.

Chapter 6 presents the design of the CompassQL visualization query language and recommendation engine, which are extensions of the Vega-Lite language in Chapter 4 and the Compass recommender engine in Chapter 5. This chapter also demonstrates how CompassQL queries can express a variety of existing chart recommendation approaches.

Chapter 7 describes the design of the Voyager 2 system, which blends manual and automated chart authoring in a single tool. This chapter also discusses how CompassQL serves as a unified representation for both specification and recommendation in Voyager 2, and reports the user study comparing Voyager 2 with PoleStar.

Finally, Chapter 8 summarizes this thesis and discusses recent developments that have been built on top of the Vega-Lite, CompassQL, and Voyager systems. This chapter also outlines new research directions for exploratory data analysis tools and visualization recommendation.

1.3 Prior Publications and Authorship

While I am the principal author of the research in this thesis, the research is also a product of years of collaboration with my primary PhD advisor, Jeffrey Heer, as well as my mentors and colleagues at the University of Washington Interactive Data Lab and Tableau Research, especially Bill Howe, Jock Mackinlay, and Dominik Moritz.

The interview study on process and challenge in exploratory data analysis (Chapter 2) was a joint work with Yang Liu who co-led the interview and analysis. Jeffrey Heer, Bill Howe, and Jock Mackinlay also provided invaluable advice and comments.

Dominik Moritz, Jeffrey Heer, and I co-authored the original version of Vega-Lite (Chapter 4) to support chart recommendation in the Voyager system in our *IEEE InfoVis'15* paper [150]. With Arvind Satyanarayan, we later extended Vega-Lite to support view composition and interactions (Section 4.2). Vega-Lite's unified grammar of interactive graphics was published at *IEEE InfoVis'16* [118]. Finally, many undergraduate students at the University of Washington and Google Summer code students including Will Strimling, Matthew Chun, Yuhan "Zoe" Lu, Youying Lin, Ayush Saraf, Lingyue "Cynthia" Zhang, Swojit Mohapatra, Akshat Shrivastava, Chanwut Kittivorawong, Sira Horradarn, Melissa Diamond, Saba Noorassa, and Souvik Sen have contributed to Vega-Lite over the years, leading to its successful adoption.

The Voyager, CompassQL, and Voyager 2 projects in Chapters 5–7 (published at *IEEE InfoVis'15* [150], *ACM SIGMOD HILDA'16* [149], and *ACM SIGCHI'17* [151] respectively) are joint work with many collaborators. Bill Howe, Anushka Anand, and Jock Mackinlay provided invaluable advice for these projects. Dominik Moritz was a significant contributor to the design and implementation of the original Voyager system and the PoleStar system that we used as a controlled condition for our user studies in Chapters 5 and 7. We also used both PoleStar and Voyager's source code as the basis for the development of Voyager 2 (Chapter 7). Zening Qu, Riley Chang, Felix Ouk helped with the implementation of the Voyager 2 and CompassQL

prototypes. Since the publication of Voyager 2 and CompassQL, Lingyue “Cynthia” Zhang, Felix Ouk, Shaheen Sharifian, Alan Banh, and Fion Chan have contributed to the enhancement and maintenance of Voyager 2, while Halden Lin has contributed to CompassQL. Our colleague at Bocoup including Jim Vallandingham, Yannick Assogba, K. Adam White, and Irene Ros have also contributed many usability improvements for the tools. Finally, Ji Zhang, Saul Shanabrook, and Brian Granger have built the Voyager 2 extension for the JupyterLab data science environment.

To reflect my collaborators’ contributions, I will use the first-person plural in these chapters.

2 Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study

Exploratory data analysis (EDA), as introduced by Tukey [135], aims to complement formal confirmatory analysis with a "flexible attitude", letting data exposure inform analysts' modeling decisions [136]. In EDA, analysts usually "explore" different aspects of data by examining data values, derived statistics, and visualizations. While Tukey's original practice mainly relied on manual calculation and hand-drawn graphics, many have since developed a number of exploration software tools and techniques (*e.g.,* [36, 45, 51, 125, 131, 144]).

Today, EDA is widely adopted as a critical part of data science, both in industrial and scientific settings [73]. Though analysts perform data exploration in various kinds of analyses, little research has observed how analysis goals and context affect the day-to-day practice and challenges of exploratory analysis. Understanding these issues might inform the design of improved data exploration tools.

To better understand current EDA practices, we conducted semi-structured interviews with 18 analysts from both academic and professional settings. We asked the analysts to describe their analysis goals, tasks they performed, and challenges they faced in their exploration.

From the interviews, we find that analysts often couple exploration with other tasks including acquisition, wrangling, and reporting. We identify two common exploration goals: *profiling* (understanding what the data contain and assessing data quality) and *discovery* (gaining new insights). Though the EDA literature emphasizes discovery, we observe that all participants engage in profiling across all of their analyses, while discovery only reliably occurs in the context of open-ended analyses, which participants perform less often. We also describe the analysts' context including tools, domain knowledge (or the lack thereof), and involved stakeholders.

Next, we discuss recurring observed challenges in EDA, and report how analysis goals and context impact them. For example, we find that analysts usually have to explore numerous variable combinations, requiring them to apply domain knowledge to select and reduce the number of variables. As analysts perform repetitive tasks, they curate analysis templates to automate their routines and help them follow best practices. While exploring data, analysts often have to switch to other tasks including data acquisition and wrangling, as well as consulting with and reporting to other stakeholders. Due to limited time, analysts may also need to move on to other tasks before completing their exploration.

Finally, we identify design opportunities for exploration tools. We argue that tools could help mitigate these observed challenges and facilitate rapid and systematic exploration by providing automation for routine tasks and guiding analysis practices. We also note a lack of support for data wrangling and navigation of analysis history within exploration tools.

2.1 Background and Related Work

We build on the exploratory data analysis literature and complement prior work on understanding data analysis.

2.1.1 Exploratory Data Analysis

Exploratory data analysis stems from the collection of work by the statistician John Tukey in the 1960s and 1970s [46, 76, 77, 135]. His seminal book [135] compiles a collection of data visualization techniques as well as robust and non-parametric statistics for data exploration. Many communities including Statistics, Human-Computer Interaction, and Information Visualization have since contributed new data exploration tools and techniques (*e.g.,* [36, 45, 51, 125, 131, 144]). (See Chapter 3 for detailed reviews of related data exploration tools.)

While Tukey did not explicitly define the goals of EDA, his and other writings about EDA [24, 29, 30, 52, 56, 69, 87, 97, 120, 140] mostly focus on the discovery of structure and patterns in the data, and consider EDA a step that precedes formal modeling or confirmatory analysis. However, some [39, 40, 145] consider that EDA also covers profiling [21, 99], or initial data examination to detect data quality issues. Some also state that EDA may occur without formal modeling [41]. Our study provides evidence that EDA goals include both profiling and discovery, though we observe that discovery only reliably occurs in open-ended analyses while all participants engage in profiling across all of their analyses. We also find that some analysts perform exploration to clean or summarize data without modeling involved. We complement the EDA literature by reporting observed EDA challenges.

2.1.2 Understanding Data Analysis

Some prior studies investigate specific problems in data analysis such as the effects of latency [88] and multiple comparisons [154]. Some research specific tools including computational notebooks [114], interactive visualizations [27, 153], and dashboards [116]. In contrast, we study the day-to-day practice of EDA, which involves many tools and challenges.

Some studies focus on data analysis for specific user groups. Kwon and Fisher [44] discuss visual analytic challenges for novices. Conversely, we study experts whose jobs involve data analysis. A few [43, 82, 105] study data analysis within intelligent agencies, which shares many challenges with our findings due to exploratory and collaborative nature of their work. However, these agencies often analyze text documents whereas our participants mostly explore structured data.

Many prior studies describe tasks observed in our study. Some discuss low-level tasks for visual exploration. Amar *et al.* [22] present a taxonomy of visual analytics tasks. A few [42,75] identify operations that analysts do to visualize data. In this study, we also discuss other tasks that analysts often couple with exploration such as data acquisition and wrangling. While prior studies [60, 80, 145] describe similar high-level tasks in the data analysis process, we complement these previous studies by characterizing exploration goals and discussing how they affect tasks and challenges in EDA.

Prior work also discusses some data analysis challenges observed in our study. Many (*e.g.,* [50, 79, 80, 83, 109]) discuss challenges for data wrangling such as data integration, data cleaning, and handling large data. Here we discuss how data wrangling couples with and impedes exploration.

For exploration challenges, Lam [86] discusses interaction costs for visualizing data such as choosing data subsets and performing repetitive physical motions. Kidd [84] observes that knowledge workers often focus on implications for decision-making rather than producing generalizable knowledge. Batch *et al.* [27] also comment that visualization tools lack integration with data science workflows.

Several prior studies [35, 58, 75, 80, 108] note the importance of analytic provenance. Ragan *et al.* [108] also characterize types and purposes of provenance in visual analytics. Some studies [74, 80, 82] identify how and why analysts collaborate, and discuss impediments

for collaboration. Card *et al.* [114] also describe the tension between exploring data and documenting insights for computational notebook users.

Though this study shares some findings with prior work, none of the prior work, to our knowledge, overviews the day-to-day process and challenges of EDA, and discuss how analysis goals affect them. None of the prior work examines how analysts determine the end of an exploration, either.

2.2 Methods

To better understand day-to-day practice of exploratory data analysis, we conducted semi-structured interviews with analysts across academia and industry.

2.2.1 Participants

We interviewed 18 participants (11 male, 7 female) from both academia and industry. Six of them were researchers from two universities and five fields including Astronomy, Oceanography, Medicine, Statistics, and Mechanical Engineering. One participant was a data analyst for a university library. The remaining participants were industry analysts from seven software companies, holding job titles including UX researcher, Data Artist, Data Scientist, Data Analyst, and Business Intelligence Engineer. These industrial analysts worked on various topics including video-gaming, safety and trust, real estate, logistics, and advertising. In this chapter, we use the term "analyst" to generally refer to any participant as all participants' jobs primarily involved data analysis.

To recruit participants, we emailed our contacts at organizations within our personal and professional networks to forward our emails to analysts in their organizations. We used

a survey to screen participants that had at least one year of data analysis experience and performed EDA at least once a month. The participants' data analysis experience varied from 1-3 years to over 10 years. Most of them performed EDA on a daily or weekly basis, with the least frequent account being biweekly. While our recruitment strategy introduced potential sampling bias in the results, our primary goal is to characterize the space of day-to-day exploratory analysis process and challenges, not to quantify how common each specific task occur. To better quantify these results, other methods such as survey can complement our findings.

2.2.2 Interview

We conducted semi-structured interviews with one interviewee at a time. Each interview lasted from 45 to 90 minutes. We interviewed analysts at their workplace when possible, and used video calls otherwise. For each interview, we began by describing the study objective, namely to understand current practice and difficulties of exploratory data analysis. We then asked open-ended questions and encouraged interviewees to describe their specific experiences such as "walk us through a recent exploratory data analysis scenario". Our questions aimed to learn about the following topics:

- What are the analysis goals and outcomes?
- What tasks do they perform during the analyses?
- What tools do they use and how do they use them?
- Who are involved stakeholders and how do the analysts interact with them?
- How do they choose parts of a dataset to explore?
- How long does an exploration take?

- How do they decide that an exploration is complete?
- What are the key challenges in exploratory analysis and how do the analysts handle them?

2.2.3 Analysis

We analyzed the interview data using an iterative coding method. Yang Liu and I independently coded all participant data. Throughout the coding process, we discussed disagreements and iteratively revised our codes to ensure consistency across coding sessions. The rest of this chapter presents the results from this analysis. We also include representative quotes from the interviews to support these results. We use P1–P18 to refer to the participants.

2.3 Analysis Process and Context

From the interview responses, we first report observed high-level tasks in the analysis process. We then categorize analysis projects based on their overarching objectives and identify two kinds of exploration goals. We also discuss analysts' context including tools, their operational and domain knowledge, and their collaboration with involved stakeholders.

2.3.1 High-Level Tasks in the Analysis Process

From the participants' responses about the tasks they performed in their analyses, we found that they commonly performed the following high-level tasks, akin to [80,145]:

Acquisition—obtaining data either by locating existing data or collecting the data themselves.

Wrangling—transforming data to have a suitable format for analysis and to handle data quality issues.

Exploration—examining data by looking at the data values as well as their metadata, statistics, and visualizations.

Modeling—building and evaluating statistical models for testing hypotheses or making predictions.

Reporting—sharing the analysis results.

Some projects might omit some tasks. Though exploration often preceded modeling, several analysts (6/18) explored data to clean or summarize data without modeling involved. Some data were also clean and did not require wrangling.

The process was iterative and coupled exploration with many tasks. Analysts regularly explored data to assess if the data were relevant during acquisition. Similarly, analysts often explored data to decide how to wrangle them. Exploration also helped analysts discover the need to collect or wrangle more data. In addition, analysts often reported exploration results to other stakeholders and gathered feedback for more exploration. While we observed less coupling between modeling and exploration, a few analysts examined training data when they observed poor modeling results.

In Sections 2.4–2.7, we discuss common challenges in these tasks and report how analysts handled these challenges. Though analysts also explored variations of models and outputs, this chapter focuses on data exploration. Thus, model diagnostic is beyond the scope of this chapter.

2.3.2 Types of Analysis Projects

We asked the interviewees about the objectives of the projects that involved exploratory analysis. We observed four common project types, with varying levels of open-endedness.

Question Answering. All analysts (18/18) reported working on answering business and research questions, so they explored the data to check data quality before answering them. Many analysts (8/18) also noted that their questions, while predetermined, were sometimes open-ended and thus required exploration to discover answers, as P14 said:

> *"A lot of my work is more long-term open-ended research questions such as: how can we characterize the health of the users on our platform?"*

Analysts often produced analysis reports in the form of written documents and presentation slides. They also sometimes built interactive dashboards.

Open-Ended Exploration. While answering specific questions was more common, several analysts (7/18) noted that they sometimes broadly explored data to summarize and look for new insights *without* a specific question. P17, a data science consultant, reported that his clients once gave him their website's data and asked *"Please just tell me about my site."* P5, an astronomer, also said:

> *"Occasionally we get data that's surprising like the universe does something we haven't seen before and a telescope caught it. Then you sit down with the data and think 'What do I do now?"'*

Akin to question answering, analysts often produced reports to describe insights from the open-exploration process.

Model Development. Many analysts (10/18) reported cases where they performed exploratory analysis to prepare for model building projects such as training machine learning models or developing new metrics and rules. Besides the models, analysts might also deliver reports, or integrate the solutions into dashboards as their project outcomes.

Data Publishing. A few analysts (3/18) explored data while cleaning datasets for publishing on shared repositories, so others could use the datasets for other analyses.

2.3.3 Exploration Goals

As analysts described why they explored data in various projects, we identified two common exploration goals:

Profiling. A common goal for all analysts (18/18) was to learn what the data contained and assess if the data were suitable for the analyses. By broadly looking at the data and their plots, analysts could learn about their shapes and distributions, and detect data quality issues such as missing data, extreme values, or inconsistent data types. They might also check specific assumptions of the data, both in terms of expectations based on domain knowledge and mathematical assumptions required for modeling. By profiling, they learned if the data were ready for the analyses or if they needed to further wrangle the data or acquire more data.

Discovery. Many analysts (13/18) also explored data to discover new insights or hypotheses, as P17 described that his exploration goal was to *"be open-minded and learn what the data could tell me."* For question answering and modeling projects, analysts might focus on developing intuitions how to answer questions or formulate models such as learning about potential relationships between variables or rankings of feature importance. Some insights also inspired

the analysts to broadly explore other relevant factors while some helped them form and investigate specific questions.

Analysts' focus on exploration goals depended on project objectives. While the EDA literature (reviewed in Section 2.1.2) mostly focuses on discovery, we observed that profiling was a more common goal. Projects with fixed questions generally centered on profiling, though surprising observations from profiling sometimes prompted analysts to investigate and discover the causes of the surprises. Meanwhile, open-ended analyses involved both goals. Analysts often first focused on profiling, and shifted their focus to discover new insights when they felt more confident about the data.

2.3.4 Analysis Tools

The interviewees reported using and switching between multiple tools throughout their analyses. A few (P1, P11, P18) were application users who usually looked at and wrangled data in spreadsheets, and visually explored data in Tableau.

The rest (15/18) were programmers who primarily used one language among Python, MATLAB, R, and SAS to analyze data. They usually plotted data with APIs such as Matplotlib [1] and ggplot2 [144]. Several of them also used computational notebooks (*e.g.,* Jupyter [102]) to keep history for repeating and revising their analyses. Some noted that they preferred exploring data via scripting instead of using graphical interfaces as they did not have to switch tools. However, the programmers switched to other tools in some cases. P6 sometimes explored data in Tableau when it could connect to the data sources. Several used spreadsheets to inspect raw data, though they rarely wrangled data in spreadsheets like the application users. Many utilized languages such as SQL and Scalding to fetch and manipulate the data. Some used Tableau [131], Google Data Studio [14], or Microsoft PowerBI [16] for reporting.

Several analysts sometimes had to use domain-specific tools. For example, P3 explored biopsy images from a 3D scan with a specialized tool. A few industrial analysts also noted that their internal data platforms had some support for data wrangling and exploration. As domain-specific tools often had limited capabilities, analysts preferred to use general-purpose tools if possible. However, data were often already in these tools and exporting data was sometimes difficult.

2.3.5 Operational and Domain Knowledge

The analysts typically needed operational and domain knowledge in their analyses. They must know where the data were stored, and how the data were collected and processed. They also needed domain expertise to interpret the data and detect errors. Since analysts usually lacked some required knowledge, they had to study more about the problem domains and consult other stakeholders.

Job roles also affected the levels of operational and domain knowledge that analysts had. We observed that the interviewees had two kinds of job roles relative to their problem domains: *domain-specific analysts* (9/18) and *consultants* (7/18), with a few (2/18) straddling both roles in different phases of their careers. In academia, most researchers focused on their research topics, but one participant was a statistician providing solutions to multiple research domains. In industry, there were both analysts embedded into product teams and consultants who served internal or external clients. As consultants typically worked with a broader set of domains, they often had less domain expertise and thus relied more on other stakeholders, as P17 said:

> *"Since I'm not embedded with the team, I don't have the domain context. In this example where I saw elevated counts in the product's telemetry, I didn't know what it meant. I could guess, but I'm not on the team, so I have no idea."*

2.3.6 Stakeholders and Collaboration

We observed that analysts collaborated with a few types of stakeholders over the course of their analysis projects.

Clients. Most analysts (13/18) reported having clients, or stakeholders who prompted them to perform the analyses and were the direct audience for the results. Some analysts were consultants who served external clients while some worked with internal clients within their organizations such as product managers or executives.

Analysts often interacted with their clients in an iterative fashion. Besides reporting the final results, analysts might share preliminary results and ask the clients for feedback such as verifying if the results matched the clients' prior knowledge, checking if the analyses aligned with the project goals, and asking if the clients had any additional questions.

Data owners. Many participants (10/18) described interacting with data engineers or database administrators who curated, processed, or stored the data prior to their analyses. Clients were also sometimes data owners, directly providing the data for the analysts. Analysts often asked the data owners to provide additional information to help them locate, clean, and understand the data since data owners had a better understanding of the format and meaning of the data as well as where the data were stored and how they were processed.

Analysis Team Members. Though the interviewees primarily analyzed data on their own, most of them (15/18) were members of analysis teams. Thus, they regularly obtained feedback from fellow analysts and supervisors before presenting to clients. Typical feedback included additional questions to explore, technical advice for analysis techniques and implementation, as well as suggestions to make the reports easier to understand for the clients. In addition, a few interviewees noted that they worked jointly with their colleagues on some projects. Two reported splitting the work so each team member could focus on an independent scope

and make progress in parallel. Another mentioned that she and her colleague independently analyzed the same data and cross-checked if they arrived at the same results.

Besides supervisors and fellow analysts, a few interviewees had colleagues with more domain expertise in their teams. P3's medical device research team had a pathologist to give opinions on tumor image analysis. P16, a data science consultant, also reported that his organization included business-oriented "solution managers", whose duties were to bridge the communication gap between the clients and technical-oriented data scientists and help them define deliverables that matched the clients' goals.

2.4 Data Acquisition

We now discuss challenges for data exploration and relevant activities. The first step is to acquire the data necessary for the analysis. All but one interviewee (17/18) reported working with existing datasets. For business analysts, most data were from product logs or customer surveys, while many researchers worked on datasets jointly collected by their research communities. Only some (5/18) had participated in data collection, either by collecting the data themselves or requesting that certain data should be collected.

When working with existing data, finding relevant data were difficult for a few reasons. First, data were often distributed. Several analysts reported that their companies used multiple data storage infrastructures. A few researchers also mentioned that their datasets were collected and published by different research organizations. Thus, analysts typically had to search for data in many places. Moreover, data sources often had insufficient data description, having uninformative column labels and none or outdated documentation. As a result, analysts had to explore all potential datasets to assess if they were relevant to their analyses.

Some analysts also consulted data owners to locate and understand the data. They often received connections to these data owners from their clients or colleagues. However, P14 noted that finding the right people to talk to could be difficult since she worked in a remote office.

Analysts also used keyword search to look for relevant datasets in their databases. However, as the same data could be named in many ways, they must use the right keywords to find the data. For some analysts, their data sources might not have convenient search capability at all. Due to this problem, P2 noted that she was building a searchable database for her organization.

For consulting analysts, their clients often provided them data. However, in some cases the provided datasets lacked the necessary information to achieve the project goals, requiring the analysts to search for more data or terminate the project if they could not find appropriate data.

2.5 Data Wrangling

We observed that analysts often coupled data wrangling with exploration. As analysts received new data, they might want to explore the data. However, data often came from many sources or had improper format and size for analysis tools. Thus, analysts had to transform the data prior to exploration. Once they explored the data, they might discover that they needed to further handle erroneous values or rescale the data. Due to this coupling, some analysts even associated exploratory analysis with data cleaning.

Akin to prior work [80], several analysts reported that they often spent more than half of their analysis time to wrangle and clean data. As exploration tools do not support some wrangling tasks, analysts often had to switch between tools throughout the analysis, requiring additional

efforts to migrate the data. We now identify commonly observed wrangling tasks that coupled with or impeded exploration.

2.5.1 Combining Multiple Datasets

Many interviewees (12/18) had to join multiple datasets or integrate similar datasets from multiple sources. Both of which presented many challenges. To understand the similarities and differences between datasets, analysts might have to profile the datasets while combining them. P6 and P10 complained that they often had to join data from over 20 tables. A few analysts who worked on data from many platforms also had to use many scripting languages to fetch the data.

One common challenge was the inconsistency between data sources. P5, an astronomer, reported that different telescopes published data using various time systems, so she spent a few days just to get the data on the same time systems before she could combine them. P11 also described joining data with different levels of granularity: *"Voting data is collected at precinct level while health data is at a state level, and population data is served at zip code level"*.

2.5.2 Dealing with Data Size

Most interviewees (15/18) had to deal with data size, which increased data processing time, impeded sharing, or even crashed their analysis tools. P14 mentioned that it took her a few days just to retrieve the data. P3 noted that it was *"extremely difficult to share a 250GB file"*. Several analysts complained that large datasets did not work in R. P11 was annoyed that his data crashed both Excel and Tableau.

The interviewees applied a few strategies to handle large datasets. Many (8/18) reduced data size by *sampling* the data. P9 and P10 noted that their challenges for sampling included

"figuring out how large of a sample size we needed and balancing how long it would take to run" as well as *"determining how to get meaningful and represenative samples"*.

Many analysts (8/18) also reduced data size by *filtering* interesting or relevant subsets based on their domain knowledge or suggestions from domain experts. P15 also applied signal processing techniques to detect signal of interests from audio data, so she could explore just the relevant data. However, analysts might not know in advance how to filter the data until they explored the data.

Some interviewees (4/18) handled large datasets by *aggregating* them. One difficulty for aggregation was deciding the level of details. For example, aggregating time series by milliseconds could make the aggregated data too large while aggregating by year might eliminate important details for the analysis. However, as analysts sometimes lacked specific questions during exploration, they might not initially know the right aggregation level and thus had to re-aggregate the data many times during the exploration.

2.5.3 Converting Data Formats and Deriving New Data

Most analysts (15/18) had to convert data into formats expected by their analysis tools. Common formatting tasks included converting file formats and character encodings as well as manipulating data layout such as splitting data columns and reshaping datasets into long formats. A common complaint was that data formatting was time-consuming. Several analysts also complained that they had to manually format spreadsheets that did not have rectangular shapes.

Besides formatting, many analysts (13/18) derived new form of data more appropriate for their analysis. Many often rescaled data by normalizing data into certain ranges (*e.g.*, 0 to 1) or applying logarithmic transformation to make them more uniformly distributed. Several applied low-pass filters or calculated moving averages to reduce noise in the data. P14 and

P17 also mentioned coding new high-level categories from the original low-level categories. As we will discuss in Section 2.6.4, analysts also often derived tabular forms of unstructured data (*e.g.,* by calculating statistics) so they could explore and analyze the new data.

2.5.4 Handling Erroneous Values

While exploring data, most analysts (13/18) had to handle data errors such as missing data and extreme values. Handling erroneous values was challenging since any decision to filter or impute the data required domain knowledge and might affect downstream analysis. Thus, analysts often explored other aspects of the data and consulted data owners before picking a filter condition or imputation method. Since they might later find that some errors were irrelevant to their analyses, analysts sometimes "piled up" errors and kept exploring until they knew the errors were important to handle.

2.6 Data Exploration

Once analysts wrangled their data to have a proper format and size, they would explore the data, which sometimes led them to acquire or wrangle more data. In this section, we first summarize observed exploration process with a focus on tabular data, the common data form for all interviewees. We then discuss exploration challenges including choosing variables, handling repetitive tasks, exploring unstructured data, and deciding the end of exploration.

2.6.1 Observed Exploration Process

Analysts usually began exploring by checking what the data contained. For tabular data, analysts would look at table headers and, if available, read documentations about the data.

After knowing what the data were about, they would choose aspects in the data to explore (or decide to stop exploring if the data were irrelevant). As we will discuss later, the analysts would reduce the number of variables if necessary.

Analysts applied various methods to examine tabular data. To profile the data, more than half of analysts (12/18) directly looked at the data values (*e.g.,* via a print command or spreadsheet software). Many (8/18) computed summary statistics such as the range and central tendencies for continuous variables and value counts for categorical variables. Most analysts (15/18) examined univariate distributions with histograms and count plots. P2 and P7 reported using box plots, while P12 used kernel density plots. Analysts sometimes wrangled a variable during exploration, *e.g.,* by filtering irrelevant and missing values or rescaling the variable.

Analysts examined multivariate distributions for both profiling and discovery goals. They often checked certain distributions to verify their assumptions and investigated why some assumptions did not hold true. If their exploration goal included discovery, they would also explore various combinations of variables to see if they could learn interesting insights. Some of these insights might inspire them to further explore other relevant aspects of the data.

All analysts employed bivariate plots including bar, line, and scatter plots. A few (3/18) used 2D histograms, frequency tables, and contour plots. Many analysts (10/18) also explored plots with more than two variables. In many cases, they encoded the third variable in a plot with colors. P4 and P16 also displayed surfaces of functions with two input variables using 3D plots. However, P16 noted it was sometimes difficult to see relationships from a 3D plot. To examine multiple variables at the same time, several used scatterplot matrices. P2 and P8 also used parallel coordinate plots. If there were too many variables, some analysts grouped variables into small batches to avoid making the scatterplot matrices too large. As we will discuss later, a few also grouped redundant variables with correlation plots.

The analysts reported that a straightforward exploration may take a few hours to a few days. However, the data were often dirty or incomplete, requiring them to acquire or wrangle

more data before they continued exploring. Moreover, analysts often had to consult and get feedback from clients or colleagues. However, these stakeholders might not be immediately available to help, so the analysts had to switch to other projects while waiting. For these reasons, exploration may take several days or even weeks.

2.6.2 How to Choose Variables to Explore?

One common challenge was choosing variables to explore. The interviewees generally reported that they were comfortable exploring datasets with up to one or a few dozen variables. However, many (13/18) had to analyze datasets with several dozens to hundreds of variables. Several (7/18) mentioned that the number of variable combinations to explore was a challenge for them. P16 complained that picking variable was *"too time-consuming"*. P2 said that *"choosing variables was harder than plotting itself."* P10 even said he *"sometimes skipped plotting if there were too many variables."*

When there were fewer variables, analysts typically examined univariate distributions of all variables and, if possible, all bivariate distributions. If there were too many combinations, analysts often tried to choose around 10–20 variables using a number of criteria. In addition, they sometimes applied dimensionality reduction techniques.

All interviewees regularly applied domain knowledge to choose variables. For profiling, they often examined variables related to their assumptions based on prior knowledge or suggestions from involved stakeholders. For question answering and modeling, analysts might explore variables they considered relevant to their questions or likely to affect the dependent variables. For open-ended exploration, analysts might wander through data based on what they found interesting. Though a common difficulty was deciding what would be interesting for the audience, several analysts noted that they often explored relationships

that might have implications for decision-making. P11 also *"drew diagrams between variables with potential relationships"* to pick variables.

Many analysts (9/18) also reported criteria for dropping variables. They often discarded variables that were parts of their datasets, but irrelevant to their analysis. As datasets often contained duplicate or similar variables, three analysts also used correlation plots to group redundant variables. For each group, they then picked a variable that was the most reliable, having no outstanding data quality issues, and the most understandable for their audience.

Several analysts (5/18) applied *statistics and modeling techniques* to select variables. Some built simple models, such as shallow decision trees or random forests, to determine important features. A couple examined variables that correlated with dependent variables. However, these approaches have some limitations. P17 noted that industrial datasets often contained duplicated variables, which might caused some of them to appear less important in the model building approach. P2 also noted that *"sometimes there were many things that too were correlated but not important"*.

Besides selecting variable subsets, several analysts (6/18) utilized *dimensionality reduction* techniques to explore large number of variables. Many used principal component analysis (PCA) and plotted the top eigenvectors. P14 also plotted data with t-SNE [89,142]. However, dimensionality reduction could lead to interpretation difficulty, as P12 noted: *"If I have a hyper-dimension that's combining 1,000 different variables, I can't explain to my audience what it means."*

2.6.3 Handling Repetitive Tasks

We found that repetitive tasks were another factor that impeded exploration for many analysts (7/18). Some commented that they often had to *"reinvent the wheel"*, performing similar tasks

in each exploration. P8 also wished for a better way to visualize multiple variables at the same time:

> *"I wish there were a tool that I can just browse through a gallery of each variable's plot. It would be awesome to just browse through each of the variable's distribution and outliers, then move on to the next one."*

Despite the abundance of guidelines for data analysis and visualization, some analysts also noted that most tools did not incorporate such knowledge or make them easily accessible. Thus, they had to manually apply the knowledge themselves. One common challenge, especially for programmers, was recalling how to run specific analysis commands. Another common complaint was the lack of good defaults in tools. P13 complained that Matplotlib often required additional customization to make plots look good. P17 was annoyed that many plotting libraries dropped null values by defaults without informing that some values were dropped.

To avoid repetitive tasks and ensure that they followed best practices, some programmers compiled templates for commands they often used. P17 even wrote a script to generate a Jupyter notebook that included basic summary plots of all variables in a given dataset and ran basic checks for data quality issues, so he could begin exploration by browsing the notebook without rewriting analysis commands every time. Though templates were useful for saving analysis time, different datasets often had their own subtleties, so analysts needed to adjust their templates based on the data.

2.6.4 Exploring Unstructured Data

While all interviewees regularly worked with structured data such as table and networks, several (7/18) sometimes analyzed unstructured data including text, audio, genomic sequences, and images. One common challenge was the lack of methods for exploring a large collection

of unstructured data. In many cases, analysts derived new forms of data and explored the new data instead. P13 and P16 computed word frequencies for text data. P2 calculated missing call rates for genomic sequences. P15 also applied signal processing techniques to extract signals of interests from audio data. However, when it was difficult to derive a new form of data, the analysts might have to sample the data instead. For example, P3 had to profile a large collection of image data by directly examining small set of samples.

2.6.5 When Does the Exploration End?

As exploratory analysis is open-ended by nature, a common challenge was deciding when the exploration should end so analysts could move on to the next tasks. When asked how they decided to end an exploration, a handful of interviewees (5/18) responded that they did not always have a definite answer if an exploration was complete. From the interviews, we found that analysts determined to end an exploration based on multiple factors including goal satisfaction, feedback from involved stakeholders, and time constraints.

Goal Satisfaction. All analysts (18/18) typically ended an exploration once they had satisfied their goal. For discovery in question answering and modeling projects, they concluded once they had an intuition on how to formulate the answers or the models. For profiling, analysts usually stopped when they had verified all assumptions and felt that they got a good sense of the data. Analysts might choose to move on if they thought they had done a sufficient job, as P17 said:

> *"If I reached a point where I no longer saw glaring issues, I'm done. It does not mean the data is clean. In fact, I know that it is not perfectly clean. However, I'm not seeing any other issues in the data, so they're small enough that I don't need to care about them."*

The analysts' confidence whether they had done a sufficient job varied based on the exploration goals. Analysts generally felt confident about profiling, as P6 noted that *"just looking at*

distributions is not that hard." However, they were sometimes less confident, especially when the data were large. P3, who profiled samples of large image collections, reported that he felt *"confident for 90% of the time"*, but sometimes worried that he might have missed important errors in the data. For discovery, analysts generally felt less confident as the goal was more open-ended by nature. P5 even revealed that she never knew if she had comprehensively explored the data when exploring a dataset she had not seen before.

Stakeholder Feedback. Since determining if they had sufficiently achieved the exploration goals could be difficult, analysts generally performed multiple rounds of exploration where they received feedback from team members and clients in between. They then used feedback from team members and clients to assess if they need to further explore the data. Some analysts described that they would stop profiling once their clients and colleagues no longer had concerns about the data. A few also noted that early feedback from colleagues sometimes helped them terminate a low-value project early and let them focus on more important projects.

For open-ended discovery, analysts often ended a round of exploration when they had shareable results. One industrial analyst (P9) mentioned that he usually stopped when he found a result *"worth sitting down and discussing."* P5 who used a large public data (P5) for her research mentioned that she stopped exploring when she *"discovered enough material [to analyze] for a paper"*. By writing a paper, she then received feedback from the research community, driving her to do further analysis. Analysts also sometimes stopped exploring if the data had nothing interesting.

Time Constraint. Many analysts (9/18) cited time limit as another major factor that prompted them to stop exploring. P16 mentioned that *"it is okay to explore the data for a couple of weeks, but after that I will need to start the other parts of the work."* P17 also described the pressing nature of his work: *"we are developing models, and we have to deliver. It's happened that we have some*

stones left unturned—sometimes we come back, sometimes we don't." For time-sensitive projects, analysts might skip some parts of the exploration, as P8 said:

> *"If I have to do it fast, I would not spend most of my time in exploratory analysis. I'll do some spot checks like just checking the ranges. I would not even look at the distributions and just go right into modeling."*

Since analysts often had limited time to explore large amount of data, it was difficult to perfectly explore all aspects of the data. Thus, analysts sometimes returned to exploration after moving on to modeling. A few interviewees reported that poor modeling results led them to further investigate if there were data quality issues that caused the problems.

2.7 Reporting and Sharing Analysis

As data analysis is iterative and collaborative, analysts had to share their analysis results throughout the process. We now discuss common challenges for sharing analysis results.

2.7.1 Adjusting Reports to Match Analysis Audience

Many analysts (12/18) described the need to adjust their reports to match their analysis goals and the audience's background. P17 mentioned that his goal was to *"produce insights for the audience with the least amount of effort for them to understand."* P18 also noted that *"explaining complicated things in a simple way"* was the hardest thing in data analysis.

We observed a few strategies for simplifying analysis reports. First, analysts typically avoided using sophisticated plots, such as box plots, in reports for stakeholders with less data analysis expertise. Moreover, while their explorations might have many delicate details, they often presented only the most important findings, such as ones that had implications for decision-

making. However, a challenge was that their analysis audience had varying degrees of expectations. Some might even expect to explore the reports themselves, requiring the analysts to create dashboards for the reports.

The need to communicate with audience also led the analysts to align their analysis decisions with the audience's background. When possible, they would choose concepts that the audience were familiar with. As discussed earlier in Section 2.6.2, one criterion to choose a variable from a group of redundant ones was whether the audience would understand it. P8 also reported that he avoided introducing a new metric in his analysis as there was a similar but widely-used metric.

Analysts also collaborated with their colleagues to adjust their presentation. For example, a consulting data scientist (P16) mentioned that he always worked with a project manager to ensure that his presentation would be understandable to the clients and directly address their analysis goals.

2.7.2 Analysis Sharing and Provenance

Sharing analysis history across organization was a common challenge for several analysts (6/18), as P14 said: *"I often felt that I'm reinventing the wheel, but it'd take me a week to find somebody who already did something similar."* A few also reported that their companies tried to use collaboration platforms such as a wiki to share analysis summaries. However, these attempts eventually got abandoned because analysts did not want the extra work to write a summary, in part because they had already presented their analysis via other forms of reports such as slides. P9 also noted the tension between doing more analysis and writing more reports:

> *"Given a fixed amount of time, do I answer more questions and go as far as I can or do I go slower and write more reports? Finding the balance is a bit hard."*

Analysts also had to revisit their own analysis history to repeat an analysis with new data, or to help them recall prior work when they summarized an analysis for reporting or switched from another project. As discussed earlier in Section 2.3.4, some analysts utilized computational notebooks to keep analysis history. However, some analysts had difficulty keeping analysis history, as P12 said:

> *"I and many other analysts I know often went through an awful lot of charts and later realized there were a few that we wanted but didn't save along the way."*

2.8 Design Opportunities

Based on these interview results, we now identify design opportunities for improving data exploration tools.

2.8.1 Facilitate Rapid Exploration with Recommendation

From the interviews, we observe many challenges in exploratory analysis that suggest opportunities to augment data exploration with automation and guidance.

First, as analysts often need to perform repetitive tasks and have limited time to explore data, tools should provide automation to help analysts focus on analyzing the data rather than executing routine tasks. While some existing tools provide sensible defaults for plotting commands [144] or help automate chart design [91], these features are not yet available in some popular analysis environments such as Python. More importantly, analysts still have to manually create charts one-by-one. As we observe that some analysts apply templates to automate chart generation and wish to browse chart without manually plotting them, tools can recommend charts for analysts to examine.

As part of the recommendation, tools can incorporate analysis practices as suggested by our participants. Since analysts should begin exploring data by examining univariate summaries of all variables [96,120], tools can suggest these plots for the analysts [150]. When an analyst plots an average of a variable, a tool may augment the plot with variability information to convey uncertainty, or suggest robust statistics such as median if there are outliers skewing the average. For large data, a tool may suggest approximate techniques such as sampling, online aggregation [54, 66], or density based plots such as histograms and binned scatterplots [37] instead of plotting all individual points.

Another key difficulty for data exploration is choosing variables to explore. We observe that analysts heavily rely on their judgment including determining what variables are interesting and deciding if they have sufficiently explored the data. One potential risk is that they may be biased to focus what they or their stakeholders are interested in. As a result, they may overlook important insights in the data. Tools can reduce this risk by suggesting analysts to explore other aspects of the data and promoting serendipitous discovery.

An important question is how to recommend data for analysts to explore. For profiling, tools may automatically detect and recommend variables with potential issues such as missing values or outliers [81]. Recommending data for discovery is more challenging as the goal is more open-ended by nature. While prior work [121, 138, 148] leverages statistics for recommendations, we find that analysts mostly pick variables based on their interpretation of semantic relationships between variables. Meanwhile, statistical properties are sometimes irrelevant. Thus, at a basic level, tools should allow analysts to steer suggestions based on their interests. An open research question is how to design an elicitation method that allows analysts to convey domain knowledge such as how the variables could influence each other. Tools can then store and use this information to recommend relevant variables. As analysts in the same organizations often explore the same datasets at different times, tools may also leverage prior analyses to learn relevancy between variables.

2.8.2 Support Iterative and Collaborative Workflows

Another key finding is the tight coupling between exploration and other tasks, requiring analysts to switch tools and migrate data. Exploration tools should either provide support for other tasks such as data wrangling [18] or tightly integrate with existing data analysis ecosystems. For example, the JupyterLab data science environment [15] has an extension system that can be used to develop an exploration tool for Jupyter Notebook users. Moreover, tools should consider using shared in-memory data format (*e.g.,* [12]) to reduce the need for data migration due to tool switching.

Another observation is the lack of support for browsing and searching history [63] in exploration tools. If analysts can efficiently search for analyses relevant to certain datasets and variables, they can better understand the data and avoid repeating existing work. Moreover, as an exploration on a dataset can be lengthy, tools should also provide interfaces to annotate important findings so that analysts can later revisit and summarize these findings for their reports. Finally, as analysts may not know if they have comprehensively explored their data, surfacing variable coverage [117] may help analysts identify unexplored directions and perform more comprehensive exploration.

2.9 Conclusion

This chapter presents the results of an interview on exploratory data analysis with 18 analysts from professional and academic settings. We characterize common exploration goals: *profiling* (assessing data quality) and *discovery* (gaining new insights). Though the EDA literature emphasizes discovery, we observe that discovery only reliably occurs in the context of open-ended analyses, whereas all participants engage in profiling across all of their analyses.

We also describe how analysis goals and context affect the tasks and challenges in exploratory data analysis. We find that analysts usually have to explore numerous variable combinations, requiring them to apply domain knowledge to select and reduce the number of variables. As analysts perform repetitive tasks, they also curate analysis templates to automate their routines and help them follow best practices. Analysts also often have to consult other stakeholders due to the lack of domain knowledge, and oscillate between exploration and other tasks, such as acquiring and wrangling additional data. In addition, analysts also report a lack of support for navigating analysis history within exploration tools.

Based on these observations, we identify a number of design opportunities for data exploration tools. We argue that tools could facilitate rapid and systematic exploration by providing automation for routine tasks and guiding analysis practices. We also note a lack of support for data wrangling and navigation of analysis history within exploration tools.

The rest of this thesis focuses on augmenting exploration with automation and guidance because this approach can mitigate many of the recurring observed challenges. As analysts primarily utilize data visualization to explore data, we present a stack of systems for augmenting exploratory data analysis with chart recommendation. The Vega-Lite visualization grammar (Chapter 4) enables a concise specification for both rapid manual chart authoring and automated chart recommendation. On top of Vega-Lite, we also use an iterative design process to develop the Voyager data exploration tools powered by chart recommendation (Chapter 5 and Chapter 7) and the CompassQL visualization query language (Chapter 6).

As the key of exploratory data analysis is the analysts' flexibility to examine different aspects of the data based on their interests, the Voyager systems augment exploration with chart recommendation while preserving user's flexibility to drive the analysis. Since we observe that analysts typically reduce the number of variables to 10-20 variables, we focus on facilitating rapid and systematic exploration for these reasonably small number of variables. As part of

the recommendation, our systems incorporate data visualization and analysis practices and also promote data discovery by enumerating plots with additional variables.

3 Related Work on Interactive Systems for Exploratory Search and Data Visualization

As this thesis aims to augment exploratory data analysis tools with chart recommendation, we draw on and extends prior research on exploratory search systems as well as tools for visualization specification and recommendation.

3.1 Exploratory Search

Exploratory data analysis [64,135] and exploratory search [93,143] share a number of characteristics: users might be unfamiliar with their resources (such as datasets), undecided about their goals, or unsure about how to reach their objectives. To perform exploratory tasks, users may either query for specific information or browse to gain an overview and discover the unexpected. As they gain new information, users may clarify their goals and engage in alternative approaches.

Exploratory search tools may employ interfaces such as faceted browsers [152] and dynamic queries [124] to let users focus on interested items. With these interfaces, users express their

intent in the form of *partial specifications* that convey criteria for desired items. For large collections, recommender systems [68] can populate a seed set to help users begin exploring or suggest relevant alternatives to selected items.

Our work attempts to support exploratory data analysis in an analogous fashion. In Chapter 5, we contribute Voyager, a faceted browser interface for statistical graphics of a single relational table. To facilitate open-ended exploration, Voyager supports navigation using facets such as the data schema, applicable data transformations, and valid visual encodings. To support smooth gradations between open-ended exploration and targeted question answering, Voyager 2 (Chapter 7) extends traditional specification interfaces [131] with two new partial view specification interfaces. First, users can apply *wildcards* to author arbitrary partial view specifications and query for views that satisfy given constraints. In addition, the system expands user's focus by recommending *related views* based on the currently specified view. To help analysts begin an exploration, both Voyager and Voyager 2 suggest univariate summaries in the initial view, promoting inspection of all data fields in the dataset.

To enable these recommendations, we also contribute Compass (Section 5.5), a visualization recommender system that produces relevant and perceptually effective views for Voyager. In Chapter 6, we also extend Compass to support generalized queries for partial view specifications in the form of CompassQL queries.

3.2 Visualization Specification Tool

Chart templates, as often found in spreadsheet applications and many visualization tools (*e.g.,* [2, 48]), are the most common way to specify visualizations as they are simple to use. However, they can be restrictive, limiting users to small sets of predefined templates and customizable parameters.

To support an expressive range of visualizations, many work instead provides languages for composing visualizations. In Semiology of Graphics [31], Bertin develops a set of vocabularies for describing data and visual encodings. Inspired by Bertin, Mackinlay's APT [90] applies formal visualization languages to generate computer-based visualizations. As we will discuss in Section 3.3, APT uses a set of compositional algebra and design criteria to automate chart design. Building on Bertin's work and APT, Wilkinson introduces the Grammar of Graphics [146], which describes basic building blocks for composing a broad range of visualizations.

Many visualization tools have since adopted this grammar-based approach. Low-level visualization grammars such as Protovis [33], D3 [34], and Vega [119] are widely used to build *explanatory* data visualization and customized analysis tools as they offer fine-grained control over the graphics. However, they require verbose specifications, impeding rapid chart authoring in exploratory analysis.

For *exploratory* visualizations, users typically prefer high-level grammars such as ggplot2 [144] as they are more concise for rapidly exploring different aspects of data. These high-level grammars enable concision by allowing users to omit low-level details and automatically providing defaults for these omitted values. To enable *interactive* specification, Tableau (formerly Polaris [131]) offers a graphical interface in which users drag-and-drop data fields onto visual encoding "shelves." These interactions produce *complete* view specifications using the VizQL high-level grammar. However, Tableau users still typically need to manually specify visualizations to explore data. In contrast, Voyager and Voyager 2 augment exploratory data analysis with chart recommendation for users to browse.

To provide a chart representation for a chart recommender system, high-level grammars are more suitable than low-level grammars, as they require fewer properties to be determined for recommendation. However, none of the existing high-level grammar is suitable for researching new user interfaces and chart recommender systems: VizQL is proprietary while

ggplot2 is embedded in the R programming environment, which is mainly designed for statistical analysis, not for developing interactive systems.

Thus, we develop the Vega-Lite visualization grammar on top of the Vega grammar [119] to provide a formal model for chart specification in Voyager and Voyager 2. Akin to Tableau's VizQL and ggplot2, Vega-Lite enables concise specification by allowing users to omit low-level details. However, unlike VizQL and ggplot2, Vega-Lite provides a JavaScript library to facilitate the development of new graphical user interfaces. In addition, Vega-Lite uses a portable JSON (JavaScript Object Notation) syntax, permitting automatic generation from a variety of programming languages.

With Vega-Lite as a chart representation, Voyager presents an alternative interaction method for exploring data by letting users browse a gallery of recommended charts instead of manually specifying charts. To support both recommendation browsing and manual specification, Voyager 2 augments a specification interface modeled after Tableau with wildcard specifications and related view recommendations. The CompassQL visualization query language is a generalization of the Vega-Lite grammar to support *partial* view specifications. By augmenting Vega-Lite with wildcards and directives for grouping and ranking recommendations, CompassQL can define an organized collection of charts rather than just a single chart, and serve as a formal model for both specification and recommendation in Voyager 2.

3.3 Visualization Recommendation

Prior systems on visualization recommendation can be classified based on whether they systems recommend *visual encodings* (*how* to visualize the data) or recommend *data queries* (*what* data to visualize).

In this section, we discuss how the systems presented in this thesis draw on and extend these prior systems. Later in Chapter 6 (Section 6.4), we also demonstrate how CompassQL queries can express many of these existing approaches for visualization recommendation.

3.3.1 Visual Encoding Recommenders

Many prior projects on visualization recommendation focus on automated chart design, or suggesting visual encodings for a set of user-specified data fields.

Inspired by Bertin's work on Semiology of Graphics [31] and Cleveland's work on graphical perception [47], Mackinlay's APT [90] proposes a compositional algebra to enumerate the space of visualizations. The system then uses a constraint logic program to codify two kinds of criteria for pruning and ranking visualizations. The *expressiveness* criteria determine whether a visualization expresses all the desired information and nothing else. The *effectiveness* criteria determine whether a visualization effectively conveys the information in a way more readily perceived than other visualizations.

Following APT, Voyager and Voyager 2 similarly apply expressiveness and effectiveness criteria to recommend visual encodings. In the underlying Compass and CompassQL recommender engines, the built-in design constraints represent the expressiveness criteria while the visual encoding rankings represent the effectiveness criteria. In Voyager 2 and CompassQL, we also allow users to directly author partial view specifications using wildcards, which are equivalent to "variables" in constraint logic programs.

Many systems also build on APT. Sage [112,113] extends APT with considerations of additional chart types based on data characteristics and integrates visualization design knowledge into graphical interfaces for chart authoring. Tableau provides Show Me features [91], which apply a set of heuristics to help analysts select mark types and encoding mappings. However, analysts still need to specify data fields they want to visualize in these systems. In contrast,

Voyager and Voyager 2 help suggest data fields and transformations based on user selection in addition to suggesting visual encodings.

Some systems [38,57,155] recommend visual encodings based on a taxonomy of predefined tasks. However, these approaches can be limited as users may not have specific tasks in mind or have multiple concurrent goals while exploring data. Thus, inferring the user's task or asking the user to select one may preempt the iterative examination process at the heart of exploratory analysis. In the absence of perfect knowledge about the user's task, Voyager and Voyager 2 suggest charts of appropriate yet diverse types that cover a variety of data variables for analysts to examine.

3.3.2 Data Query Recommenders

Prior work has also investigated statistical techniques to aid data field selection for a fixed set of encoding templates. The Rank-by-Feature Framework [121] orders histograms and scatterplots based on user-selected metrics. SeeDB [138] calculates deviation scores between data subsets to recommend aggregate views. Quality metrics [32] and scagnostics-based techniques (*e.g.,* [23,147]) can also help identify interesting data fields. However, the use of fixed visualization templates limits the utility of these tools for exploratory analysis. In contrast, the recommendations in Voyager and Voyager 2 span data fields, transformations, *and* visual encodings.

In Chapter 2, we also observe that analysts typically select data fields based on domain knowledge instead of using statistical properties, which are sometimes irrelevant. Thus, our implementation of Voyager and Voyager 2 do not focus on ranking data fields based on statistics, though their interfaces and the CompassQL framework can be extended to support statistics-based rankings.

3.3.3 Hybrid Recommenders

While the systems described above recommend either visual encodings or data queries, many recent systems present chart galleries that vary both data fields and encodings.

Autovis [148] generates charts based on input data to facilitate initial data exploration. VizDeck [104] presents a gallery of recommended charts based on statistical properties of interest. The system also introduces a voting mechanism by which users can adjust the ranking and supports keyword queries to search for charts. However, both AutoVis and VizDeck provide limited user control for specifying data fields and visual encodings, thus restricting focused exploration.

Inspired by the Design Galleries project [94], Van den Elzen [137] introduces a system for browsing small multiples that vary the selected parameter type. Zenvisage [126] presents a chart gallery based on queried visual patterns. These systems allow users to explore a local neighborhood of the visualization specification space, but do not facilitate breadth-oriented exploration of different variable combinations. In contrast, Voyager presents both data variations and design variations in a browseable gallery and provides navigational facets based on data fields and transformations to aid broader data exploration. To facilitate smooth gradations between open-ended exploration and focused question answering, Voyager 2 augments traditional chart authoring tool with two types of recommendation interfaces. *Related views* suggest charts relevant to the user's focus without requiring users to select a parameter type. *Wildcards* then enable users to generate multiple views that align with their analysis goals by enumerating selected parameters.

4 Vega-Lite: Representation for Specifying and Reasoning about Charts

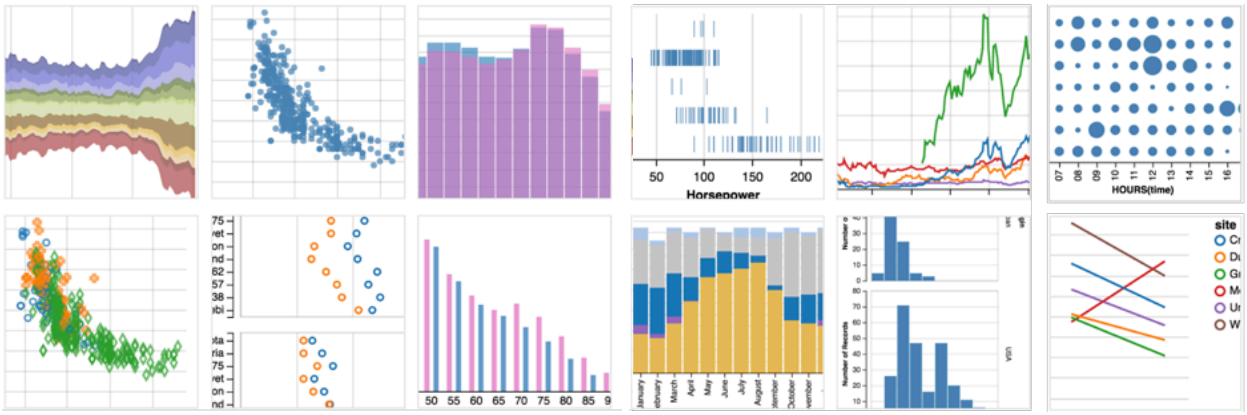


Figure 4.1: A variety of visualizations supported by Vega-Lite *unit specifications*, the basic form of Vega-Lite for describing Cartesian plots.

To augment exploratory data analysis tools with chart recommendation, we first need a formal model for specifying and reasoning about charts in our recommender systems.

As discussed in Chapter 3, high-level visualization grammars offer concise specification appropriate for programmatic recommendation. However, none of the existing high-level grammars is suitable for researching new user interfaces and chart recommender systems.

The VizQL formalism underlying Tableau is proprietary while ggplot2 is embedded in the R, which is mainly designed for statistical analysis, not developing interactive systems.

In response, we present the *Vega-Lite* visualization grammar as a formal model for representing visualizations in our user interfaces and recommender systems. Akin to Tableau's VizQL and ggplot2, Vega-Lite enables concise specification by allowing users to omit low-level details. The Vega-Lite compiler then uses a rule-based system to infer sensible default values for the omitted properties and translate a Vega-Lite specification into a detailed, low-level Vega specification [119].

To facilitate the development of new graphical user interfaces and chart recommenders, Vega-Lite provides a JavaScript library and uses a portable JSON (JavaScript Object Notation) syntax, permitting generation from a variety of programming languages. With a concise JSON syntax, Vega-Lite can also support rapid manual authoring and serve as a file format for sharing visualizations. By building on top of Vega, Vega-Lite leverages Vega's performance and flexibility across platform, supporting both browser-side and server-side rendering via SVG or Canvas.

Next, we will describe the basic form of Vega-Lite specifications. We will then briefly describe the view composition and interaction extension to the Vega-Lite grammar. Finally, we discuss new applications that Vega-Lite has enabled beyond the chart recommender systems and interfaces in this thesis.

4.1 Basic Specification in Vega-Lite

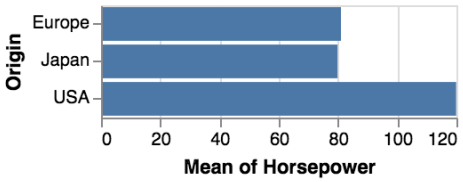
The simplest form of a Vega-Lite specification is a unit specification, which defines a single Cartesian plot. As a grammar, Vega-Lite provides primitive building blocks for composing an expressive range of charts. The basic building blocks for a unit specification include a

backing `data` set, a graphical `mark` type, and a set of one or more `encoding` mappings between visual `channels` (*e.g.*, `x`, `y`, `color`, etc.) and (potentially `transformed`) data `fields` or constant `values`. Figure 4.2 shows a unit specification of an aggregated bar chart in Vega-Lite.

The `data` definition identifies a data source, a relational table consisting of data records (rows) with named fields (columns). For example, in Figure 4.2 the data are from a URL (`data/cars.json`). The data table can be subject to a set of `transforms` such as `filtering` and adding derived fields via formulas (`calculate`).

The `mark` type specifies the geometric object used to visually encode the data records. Legal values include `bar`, `line`, `area`, `text`, plotting symbols (`point` and `tick`) and `rule` (for reference lines and line segments).

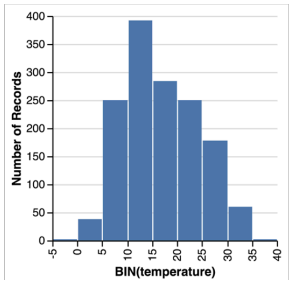
The `encoding` property represents a key-value mappings between visual encoding channels and channel definitions, which describe encoded data fields or constant values. Available visual encoding `channels` include spatial position (`x` and `y`), `color`, `shape`, `size`, and `text`. An `order` channel controls the sorting order of the graphical marks: the sequence in which points of a line or area mark are connected to each other or sorting of stacked elements (*e.g.*, for stacked bar charts and the layering order of line charts). A `detail` channel includes




```
{
  "data": {"url": "data/cars.json"},
  "mark": "bar",
  "encoding": {
    "x": {
      "aggregate": "mean",
      "field": "Horsepower",
      "type": "quantitative"
    },
    "y": {
      "field": "Origin",
      "type": "nominal"
    }
  }
}
```

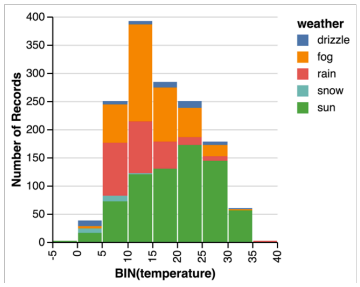
Figure 4.2: A Vega-Lite specification for an aggregated bar chart.

Histogram = (Bar with x=binned field, y=count)



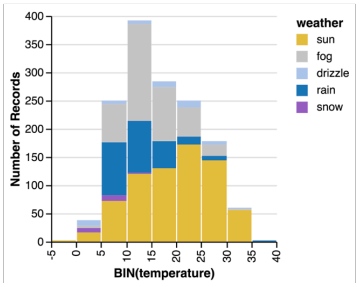

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```

Histogram + Color = Stacked Histogram




```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
      field: "weather",
      type: "nominal"
    }
  }
}
```

Stacked Histogram with Customized Colors




```
{
  ...
  encoding: {
    ...
    color: {
      field: "weather",
      type: "nominal",
      "scale": {
        "domain": ["sun","fog","drizzle","rain","snow"],
        "range": ["#e7ba52","#c7c7c7","#aec7e8",
                  "#1f77b4","#9467bd"]
      }
    }
  }
}
```

Figure 4.3: Building blocks in Vega-Lite enable composition and iterative refinement of complex graphics. (Top) A histogram is a set of `bar` `marks` that use `x` and `y` channels to encode a `bin`ned field and an `aggregate`d `count` respectively. (Middle) These building blocks also enable iterative refinement. Adding a `color` encoding to the histogram produces a stacked histogram. (Bottom) Overriding the default `scale` `domain` and `range` properties customizes the color palette.

Histogram + Column = Trellis Histogram

```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    ...
    column: {field: "weather", type: "nominal"}
  }
}
```

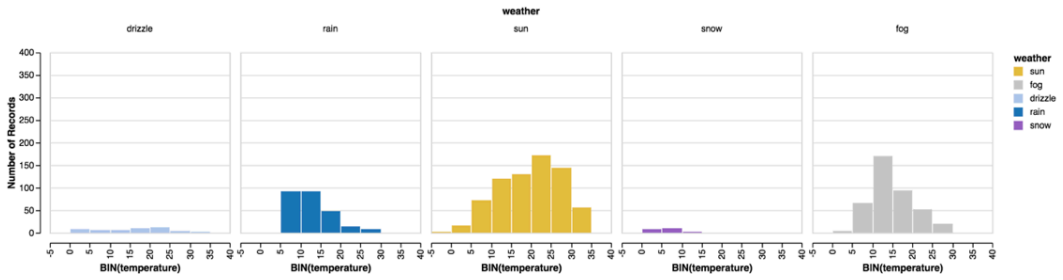


Figure 4.4: Changing the `color` channel in Figure 4.3 (bottom) to `column` produces a Trellis histogram.

additional group-by fields in aggregate plots. Finally, facet channels (`row`, `column`) split plots into small multiples or trellis plots [28,133].

A channel definition can denote a data `field` (a data attribute to visualize), along with a given data `type` [130] (one of `nominal`, `ordinal`, `quantitative` or `temporal`). Alternatively, a channel definition can specify a constant literal `value`. The data field can additionally be transformed using functions include aggregation (`aggregate`), `bin`ning, `sort`ing, `stack`ing and unit conversion for time fields (`timeUnit`). For example, the bar chart in Figure 4.2 `aggregate`s the `mean` of `Horsepower` on the x-axis.

A channel field definition may also specify other low-level details including properties of a `scale` that maps from the data domain to a visual range, and a `guide` (axis or legend) for visualizing the scale. If not specified, Vega-Lite will automatically populate default properties based on the `channel` and the data `type`. For `x` and `y` channels, either a linear scale (for quantitative data) or a discrete scale (for ordinal and nominal data) is instantiated, along with

an axis. For `color`, `size`, and `shape` channels, suitable palettes and legends are generated. For example, a quantitative color encoding automatically uses a single-hue luminance ramp, while a nominal color encoding uses a categorical palette with varied hues (Figure 4.3, middle). Users can override these default values to customize their charts. For example, Figure 4.3 (bottom) customizes the color palette by setting the `domain` and `range` properties of the `scale`.

With these basic building blocks, Vega-Lite's unit specifications can express a variety of common, useful plots of both raw and aggregated data such as area charts, scatter plots, bar charts, strip plots, line graphs, heatmaps, and variants of Trellis plots (Figure 4.1) These building blocks also enable iterative refinement. For example, we can add a `color` encoding to the histogram in Figure 4.3 (Top) to produce a stacked histogram (Figure 4.3-Middle) or instead add a `column` encoding to produce a Trellis histogram (Figure 4.4).

4.2 View Composition and Interaction Extension

Beyond unit specifications, my colleagues and I have extended Vega-Lite to support interactive, multi-view graphics [118]. A novel *view algebra* enables hierarchical composition of layered and multi-view plots via operators including *facet*, *layer*, *concatenation*, and *repeat*. Interactions can be defined by specifying and applying *selections*, an abstraction that defines input event processing, points of interest, and a predicate function for inclusion testing. With these building blocks, Vega-Lite enables concise specification of interactive multi-view plots. For example, coordinated histograms in Figure 4.5 can be defined within a few dozen lines of JSON, compared to at least a few hundred lines of code in other libraries like Vega and D3.

Detailed descriptions about Vega-Lite's grammar for view composition and interaction are beyond the scope of this thesis. For more details, please refer to our IEEE VIS'17 paper [118].

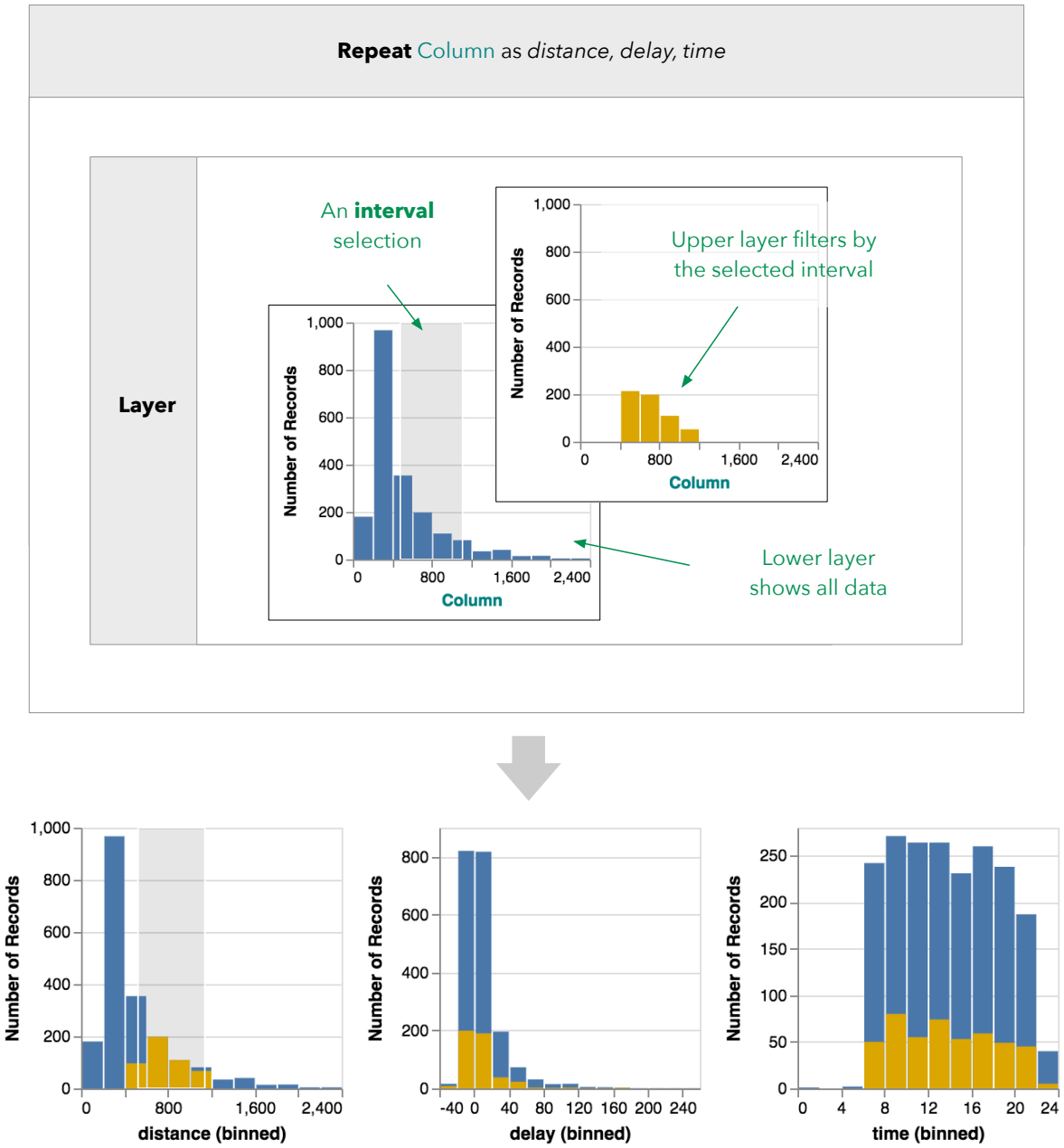


Figure 4.5: A column-based repetition of a layered histogram with a selection produces coordinated histograms. For each subplot, the lower layer (blue) shows the full dataset with an interval selection on x-axis while the upper layer (gold) shows the data in the selected range.

4.3 Applications of Vega-Lite

In this thesis, we use Vega-Lite as the representation for chart specification and recommendation. Both Voyager (Chapter 5) and Voyager 2 (Chapter 7) use Vega-Lite as a model to generate recommended charts. The CompassQL query language (Chapter 6) combines partial Vega-Lite specifications with definitions for recommendation methods to describe chart recommendation queries. We also use Vega-Lite to develop the PoleStar chart authoring tool (Section 5.6) as a baseline system for our Voyager and Voyager user studies.

Beyond this thesis, Vega-Lite has also enabled a number of applications and research projects. With a concise syntax, Vega-Lite is widely used for manual chart authoring. The Observable Javascript notebook platform has featured Vega-Lite on its tutorial [13]. Nature, a leading academic journal, mentioned that modern tools like Vega-Lite *"make scientific data more accessible and reproducible"* [103]. Vega-Lite is also used for teaching in a book for practitioners [53] and in classes at leading institutions including Stanford, Carnegie Mellon, the University of Maryland, and the University of Washington.

Vega-Lite's declarative format also enables sharing across applications and platforms. Both Vega and Vega-Lite are included as the official plotting formats in the JupyterLab data science environment. In addition, a number third-party bindings have been created for programming environments including Python [11], Elm [139], R [6, 8], Scala [9], Julia [7], and Clojure [10]. The feedback from the Python community has been positive. The developers of Altair, a popular wrapper of Vega-Lite in Python, called Vega-lite and Vega *"perhaps the best existing candidate for a principled lingua franca of visualization"*. Another widely shared review of Python visualization libraries [20] commented that *"it is this type of 1:1:1 mapping between thinking, code, and visualization that is my favourite thing about Altair [and the underlying Vega-Lite]"*.

Vega-Lite has also enabled a number of other research projects beyond this dissertation. Our lab used Vega-Lite to develop an automatic model to reason about visualization similarity and

sequencing [85] as well as a chart representation for a chart reverse engineering project [106] Our colleagues at Stanford [5] and Georgia Tech are using Vega-Lite to build natural language interfaces for data visualization and analysis.

4.4 Conclusion

This chapter presents the Vega-Lite visualization grammar, a formal model that we use to represent charts in the user interfaces and recommender systems within this thesis. Influenced by prior high-level visualization grammars for exploratory analysis, Vega-Lite offers a concise building blocks for composing an expressive range of visualizations. Beyond facilitating chart specification and recommendations in this thesis, Vega-Lite has also enabled a number of new research projects and industry applications. Vega-Lite is an open source system available at `https://vega.github.io/vega-lite/`.

5 Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations

As discussed in Chapter 2, analysts often need to perform repetitive tasks to explore a large number of variable combinations. However, existing visualization tools typically require manual specification of views: analysts must select data variables and then choose which transformations and visual encodings to apply. These decisions often involve both domain and visualization design expertise, and may impose a tedious specification process that impedes exploration. In this chapter, we seek to complement manual chart construction with interactive navigation of a gallery of automatically-generated visualizations.

We present *Voyager*, a mixed-initiative system that couples faceted browsing with visualization recommendation to support exploration of multivariate, tabular data. Voyager exchanges specification for browsing, providing an organized display of recommended visualizations and enabling user input for both chart refinement and recommendation steering. To enable breadth-oriented exploration, Voyager privileges *data variation* (different variable selections and transformations) over *design variation* (different visual encodings of the same data). Underlying Voyager is the *Compass* recommendation engine, which enumerates, clusters and ranks visualizations according to both data properties and perceptual principles. Both Voyager

and Compass represent visualizations using the Vega-Lite visualization grammar described in Chapter 4.

In this chapter, we first motivate the design of Voyager with a usage scenario. We then introduce Voyager's design considerations, interface design, and system architecture. We also describe the design of the Compass visualization recommender engine underlying the Voyager interface. To evaluate Voyager, we present a controlled user study focused on exploratory analysis of previously unseen data. We compare Voyager with PoleStar, a state-of-the-art view specification tool modeled on Tableau. Through analysis of both user performance and preference ratings, we find that Voyager better facilitates initial exploration and leads to increased data variable coverage, while PoleStar is preferable for answering more specific questions.

5.1 Usage Scenario

We first motivate the design of Voyager with a usage scenario. We illustrate how an analyst can use the system to examine data about cars [67]. The dataset contains 406 rows (cars) and 9 columns (fields).

Upon loading the data, the analyst examines the list of fields in the schema panel and their univariate summaries in the main gallery (Figure 5.2). Starting from the top left, she observes that most of the cars have 4, 6, or 8 *cylinders* (Figure 5.2-A). Using the toggle button ( ) to sort the *name* histogram by *number of records*, she notices that Ford Pinto has the highest frequency, with 6 records (Figure 5.2-B). The majority of the cars are from *origin* A (coded information, Figure 5.2-C) and the *years* 1970-1982 (Figure 5.2-D). Most of the quantitative fields appear to have log-normal distributions except for *acceleration*, which looks normally distributed (Figure 5.2-E).

Figure 5.1: Voyager: a recommendation-powered visualization browser. The schema panel (left) lists data fields selectable by users. The main gallery (right) presents suggested visualizations of different field subsets and transformations in two sections. The *exact match* section on the top contains charts with all and only the selected fields. The *suggestion* section on the bottom contains charst with all selected fields and one extra field to encourage further exploration.

Intrigued by *horsepower*, the analyst clicks the field in the schema panel. The system in turn updates the gallery with relevant visualizations (Figure 5.3). The *exact match* section (Figure 5.3-A) lists charts with varied transformations of *horsepower*. The analyst inspects the dot plot of *horsepower* and hovers over the maxima (Figure 5.3-E) to discover that the car with highest *horsepower* is a Pontiac Grand Prix. She then glances at the *suggestion* section (Figure 5.3-B), which shows charts with additional fields. She notices a correlation between *horsepower* and *cylinder*, and bookmarks the view so she can revisit it for targeted question answering after she completes her initial exploration.

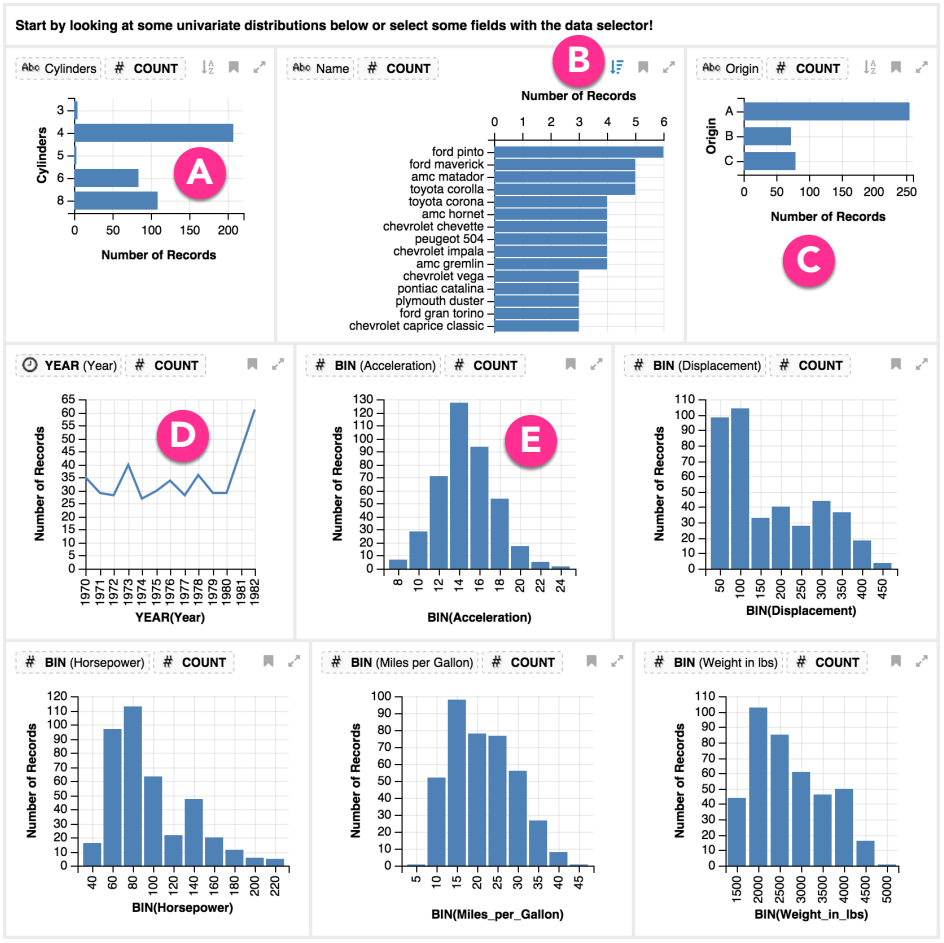


Figure 5.2: The main gallery shows univariate summaries upon loading.

The analyst wonders if other fields might be correlated with both *horsepower* and *cylinder*, so she selects *cylinder* in the schema panel. The display updates as shown in Figure 5.1. Looking at the first view in the suggestion section (Figure 5.1, leftmost view in the bottom section), she sees that *acceleration* is correlated with both fields. The analyst would like to see other ways to visualize these three fields, so she clicks the view's expand button (↙↗). This action opens the expanded gallery (Figure 5.4), which shows different encodings of the same data. She selects a small multiple view grouped by *cylinder* (Figure 5.4–B), so she can easily spot outliers in each group (Figure 5.5).

Figure 5.3: Selecting *horsepower* updates the main gallery. (A) The *exact match* section shows different transformations for *horsepower*. (B) The *suggestion* section shows charts with suggested fields in addition to *horsepower*. (C–D) Each section's header bar describes its member views. (E) Hovering over a point reveals a tooltip with more information.

At this point, the analyst wants to explore other parts of the data. She clicks the reset button to clear the selection and starts selecting new fields of interest to look at relevant visualizations. As her exploration proceeds, she bookmarks interesting views for future investigation in the bookmark gallery (Figure 5.6).

5.2 Design Considerations

While creating Voyager we faced many design decisions. The interface should not overwhelm users, yet must enable them to rapidly browse collections of visualizations with minimal cog-

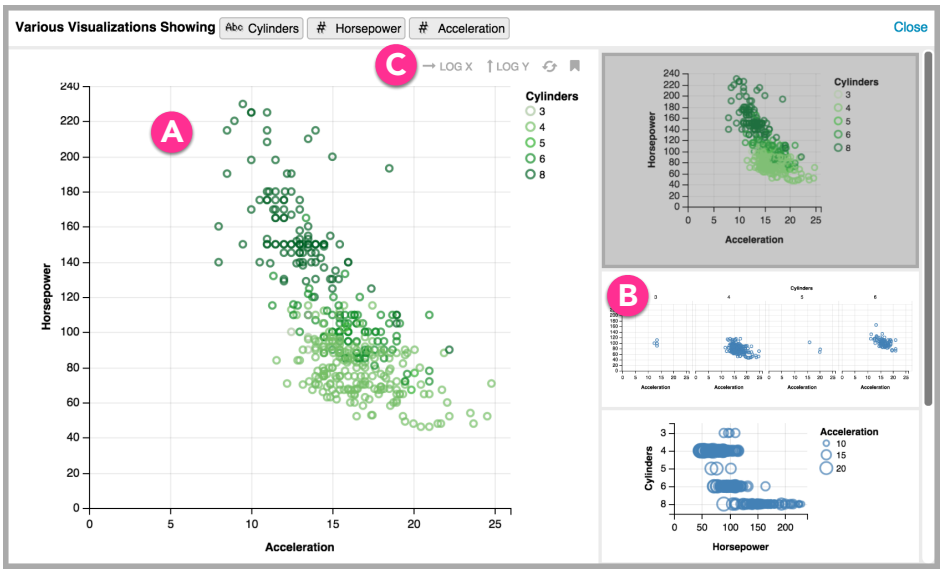


Figure 5.4: The expanded gallery for *cylinder*, *horsepower*, and *acceleration*. (A) The main panel presents the selected chart in an enlarged view. (B) The sidebar shows alternative encodings for the expanded data. (C) Controls for interactive refinement such as transposing axes and sorting nominal or ordinal dimensions.

nitive load. To guide our process, we developed a set of considerations to inform visualization recommendation and browsing. These considerations were informed by existing principles for visualization design [90], exploratory search [62,143], and mixed-initiative systems [71], then refined through our experiences across multiple design iterations.

C1. Show data variation, not design variation. We adapt this well-known maxim from Tufte [133] to the context of visualization galleries. To encourage breadth-oriented exploration [96], Voyager prioritizes showing *data variation* (different fields and transformations) over *design variation* (different encodings of the same data). To discourage premature fixation and avoid the problem of "empty results" [62], Voyager shows univariate summaries of all fields prior to user interaction. Once users make selections, it suggests additional fields beyond those explicitly selected. To help users stay oriented, avoid combinatorial explosion,

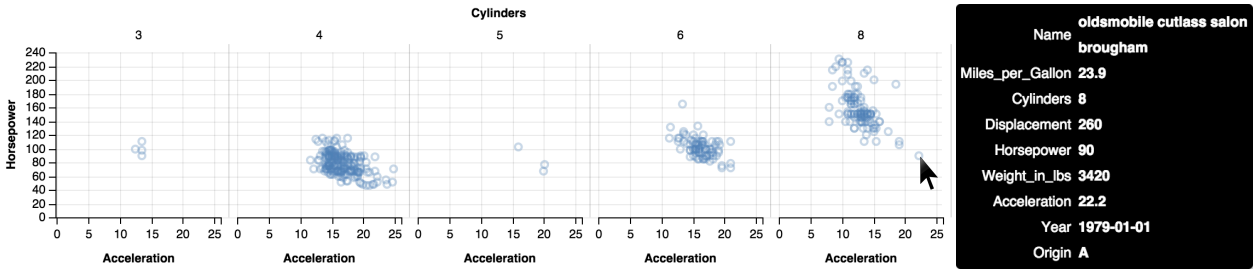


Figure 5.5: Scatter plots of *horsepower* vs. *acceleration*, partitioned by *cylinder*. An analyst hovers the mouse over an outlier to view details-on-demand.

and reduce the risk of irrelevant displays, Voyager currently "looks ahead" by only one field at a time.

C2. Allow interactive steering to drive recommendations. Analysts' interests will evolve as they browse their data, and so the gallery must be adaptable to more focused explorations. To steer the recommendation engine, Voyager provides facet controls with which analysts can indicate those fields and transformations they wish to include.

C3. Use expressive and effective visual encodings. Inspired by prior work on automatic visualization design [90, 91], Voyager prevents misleading encodings by using a set of *expressiveness* criteria and ranks encodings based on perceptual *effectiveness* metrics [31, 47].

C4. Promote reading of multiple charts in context. Browsing multiple visualizations is a complex cognitive process, arguably more so than image or product search. We must consider not only the comprehension of charts in isolation, but also in aggregate. When possible, Voyager consistently orders related charts such that effort spent interpreting one chart can aid interpretation of the next. For example, Voyager aligns axis positions and uses consistent colors for fields (Figure 5.1). Voyager organizes suggested charts by clustering encoding variations of the same data and showing a single top-ranked exemplar of each cluster. If desired, users can drill-down to browse varied encodings of the data. Voyager also partitions

the main gallery into a section that involves only user-selected fields and a section that includes additional (non-selected) fields recommended by the system.

C5. Prefer fine-tuning to exhaustive enumeration. Even a simple chart might have a number of important variations, including the choice of sort order, aspect ratio, or scale transform (*e.g.,* linear vs. log). Rather than using up space in the gallery with highly similar designs, Voyager collapses this space of options to a single chart with default parameters, but supports simple interactions to enable fine-tuning.

C6. Enable revisitation and follow-up analysis. Successful explorations may result in a number of insights worthy of further study. Exploratory tools should assist the transition to other stages of analysis. Voyager provides a bookmarking mechanism to allow analysts to revisit interesting views or to share them with collaborators. By representing all visualizations in a high-level grammar (*Vega-Lite*), Voyager can easily export visualizations for publishing or sharing with other tools.

5.3 The Voyager Interface Design

Voyager's interface (Figure 5.1) consists of a schema panel (left) and a visualization gallery (right). Analysts can select fields and desired transformations in the schema panel; these selections become input for the recommendation algorithm. The main gallery presents recommended visualizations. Each chart supports interactive refinement, bookmarks, and expansion to increase the chart size and see related views. Undo buttons are provided in the top panel (Figure 5.1, top).

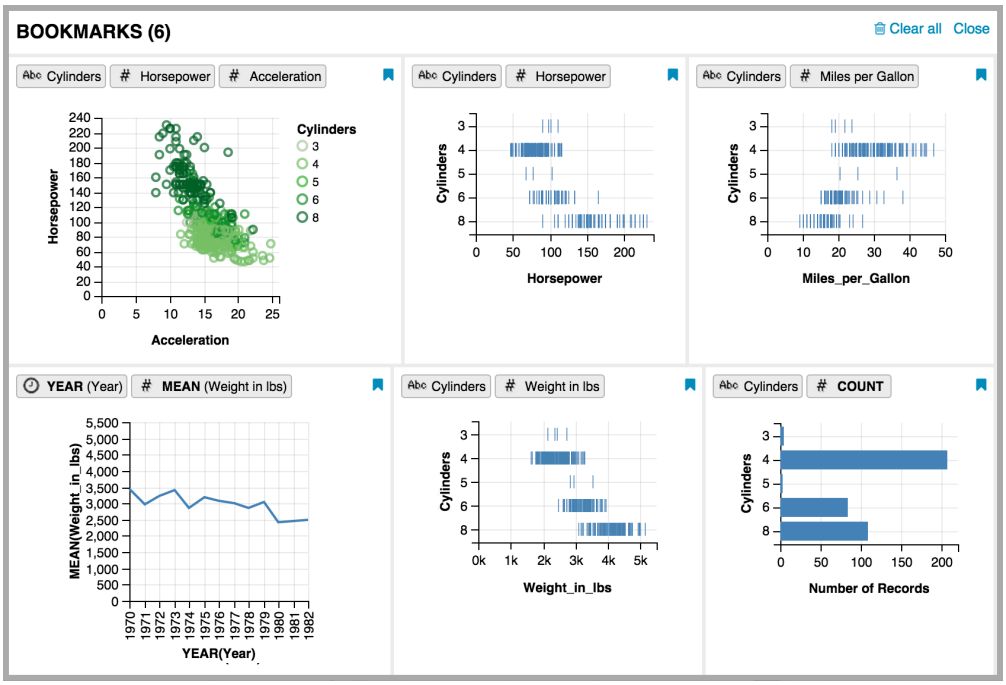


Figure 5.6: A bookmark gallery of visualizations saved by an analyst.

5.3.1 The Schema Panel

The schema panel (Figure 5.1, left) presents a list of all fields in the data table. By default the list is ordered by data type and then alphabetically. For each field, the schema panel shows the following items from left to right: (1) a checkbox representing inclusion of the field in the recommendation, (2) a caret button ▾ for showing a popup panel for selecting transformations, (3) a data type icon, (4) field name and function, (5) and a basic information button ⓘ, which upon hover shows descriptive statistics and samples in a tooltip.

To steer the recommendations (C2), users can click a field to toggle its inclusion or can select transformation functions in the popup panel revealed by clicking the caret. Selected fields are also highlighted with a surrounding capsule. Similar capsules are used in the gallery to facilitate comparison (C4). Data transformation functions are indicated using bold capitalized text (*e.g.*, **MEAN**).

5.3.2 The Main Gallery: Browsing Recommendations

The main gallery presents views that represent different data subsets relevant to the selected fields. To prioritize *data variation* over *design variation* (C1), each view in the main gallery shows the top-ranked encoding for each unique set of fields and transformations. To help provide meaningful groups (C4), the gallery is divided into two sections: *exact match* and *suggestion*. The top of each section (Figure 5.3, C–D) contains a header bar that provides a description of its member views. The exact match section (Figure 5.3-A) presents views that include only selected fields. In contrast, the suggestion section (Figure 5.3-B) includes suggested fields in addition to selected fields. If the user has not selected any fields (as in Figure 5.2), only the suggestion section is shown, populated with univariate summaries (C1).

Each entry in the gallery contains an interactive visualization. The top of each view lists its member fields in capsules. The capsules for user-selected fields (solid border, darker background, Figure 5.7-A) are visually differentiated from capsules for suggested fields (dashed border, lighter background, Figure 5.7-B). The top right of each view (Figure 5.7-C) contains *bookmark* and *expand view* buttons. During exploration, analysts can bookmark views they wish to share or revisit (C6); bookmarked visualizations can be viewed in the bookmark gallery (Figure 5.6). Analysts can also hover over data points to view details-on-demand (Figure 5.5).

Figure 5.7: The top of each view shows user-selected fields (a), suggested fields (b), and buttons to bookmark or expand the view (c).

Voyager attempts to parameterize and layout charts such that reading one chart facilitates reading of subsequent related charts (C4). To do so, Voyager places charts with shared axes in close proximity to each other. Moreover, Voyager suggests the same visual encoding (axis position, sorting, spacing, palettes, etc.) for the same field to aid scanning and reduce visual clutter. For example, it uses a consistent color palette for *cylinders* and aligns y-axes for *cylinders* and *horsepower* in Figure 5.1). To further aid comparison, all capsules for the same field are highlighted when the user hovers over a capsule.

5.3.3 The Expanded Gallery: Inspecting Alternative Encodings

An analyst can click a chart's *expand view* button to invoke the expanded gallery (Figure 5.4). This mode allows analysts to interact with a larger visualization and examine alternative visual encodings of the same data. The top-right corner of the main panel (Figure 5.4-C) includes controls for interactive refinement (C5): transposing axes, sorting nominal or ordinal dimensions, and adjusting scales (*e.g.,* between linear and log). Thumbnails of alternative encodings are presented in a sidebar. Analysts can click a thumbnail to load the chart in the main panel.

5.4 The Voyager System Architecture

Figure 5.8 depicts the relationships between the major system components. Voyager's browser interface displays visualizations and supports user navigation and interaction. The *Compass* recommendation engine takes user selections, the data schema, and statistical properties as input. It then produces recommended charts by clustering them based on data and visual similarity and ranking them by by perceptual effectiveness heuristics. The output are in the form of Vega-Lite specifications (Chapter 4, which are then compiled into detailed *Vega* [119] specifications and rendered using the Vega Runtime APIs.

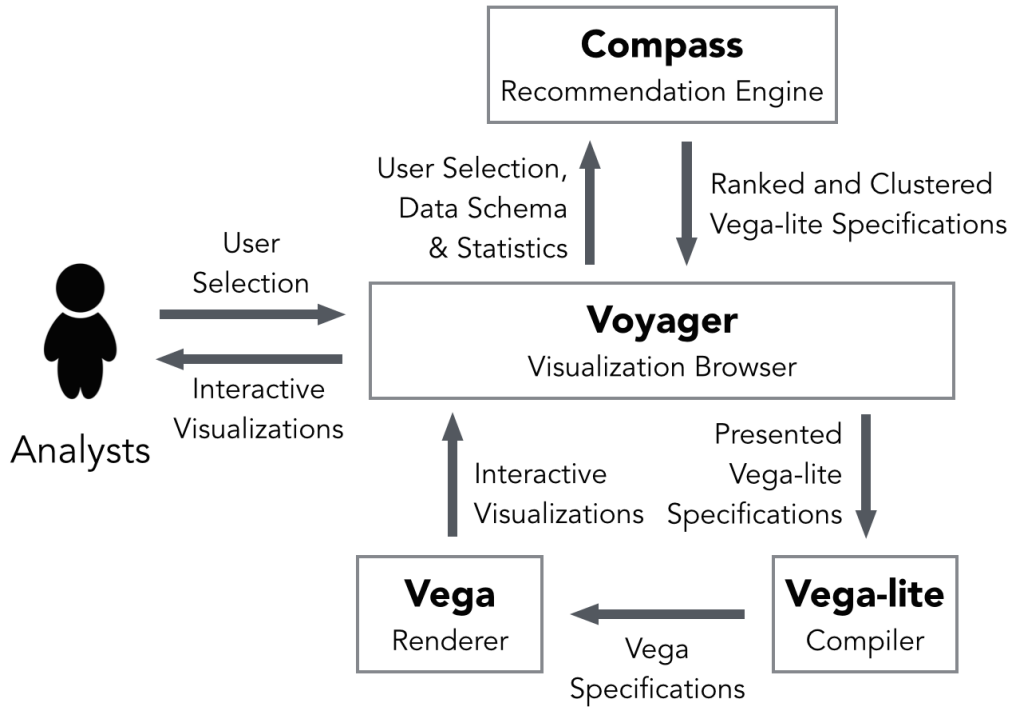


Figure 5.8: Voyager's system architecture. Voyager uses Compass to generate clustered and ranked Vega-Lite specifications. These specifications are translated to Vega and rendered in the Voyager interface.

5.5 The Compass Visualization Recommender Engine

We now describe the Compass recommender engine underlying the Voyager interface. The goal of Compass is to support rapid, open-ended exploration in Voyager. Compass generates an expressive set of visualization designs (C3), prunes the space of recommendations based on user selection (C2), and clusters results into meaningful groups (C4).

Compass takes the following input: (1) the data schema, which contains a set of fields (D); (2) descriptive statistics for each field including cardinality, min, max, standard deviation, and skew; (3) the user selection, which consists of a set of selected fields ($U \subset D$), preferred transformations for each field, and a set of excluded fields.

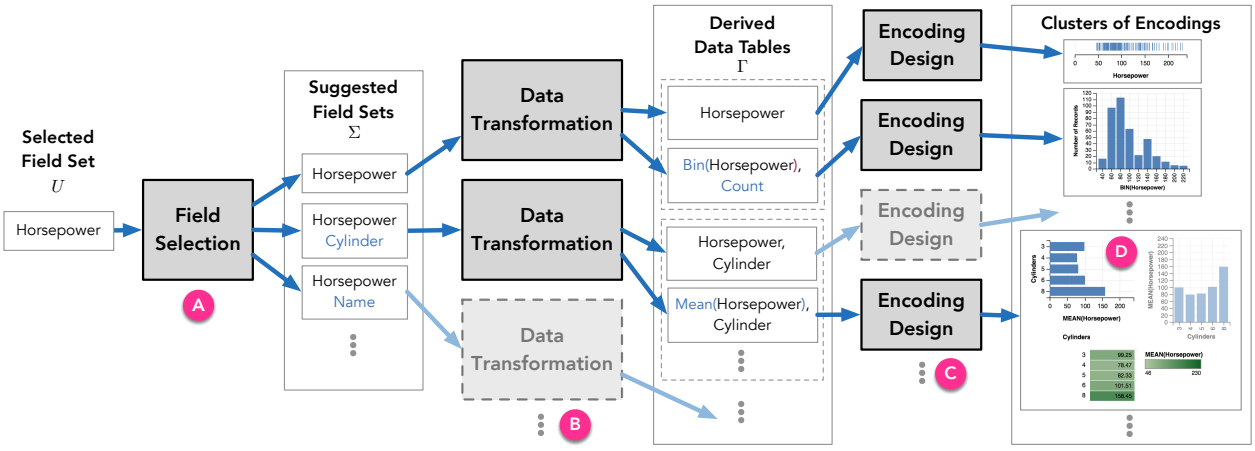


Figure 5.9: Compass's 3-phase recommendation engine. (A) *Field selection* takes user-selected field sets and suggests additional fields (B) *Data transformation* applies functions including aggregation and binning to produce data tables for each field set. (C) *Encoding design* generates visual encodings for each data table, ranks results by perceptual effectiveness score, and prunes visually similar results.

Compass enumerates, ranks and prunes recommendations in three phases, taking output from each phase as input to the next phase. The process is depicted in Figure 5.9. First, Compass *selects fields* by taking user-selected field sets and suggesting additional fields. It then *applies data transformations*, including aggregation and binning, to produce a set of derived data tables. For each data table, it *designs encodings* based on *expressiveness* and *effectiveness* criteria (C3) and prunes visually similar results to avoid exhaustive enumeration (C5). The multiple phases of pruning and ranking allow Compass to constrain the search space early in the generation process, and to produce clusters of visualizations that are grouped by their underlying data tables.

The design of Compass is admittedly modest. Though more advanced recommender systems are possible, the primary goal of Compass is to develop and evaluate an overall approach to breadth-oriented data exploration. In lieu of more sophisticated methods, we intentionally limit ourselves to "single-step" field additions and interpretable, deterministic heuristics for

pruning and ranking. We view the design and evaluation of improved recommenders (likely expressible within the current Compass architecture) as important future research.

5.5.1 Selecting Fields

Compass first suggests fields beyond what the user has explicitly selected, producing new field sets for encoding. The primary goals of this phase are to recommend additional variables that the analyst might otherwise overlook (C1) and to avoid the "empty results" problem [62].

Prior to user selection, Compass suggests a univariate summary of each field. When a user selects a variable set U (of size $|U| = k$), Compass returns a sequence of field sets $\Sigma = [U, V_1, V_2, \ldots, V_n]$, where U is the original user selection, $n = |D - U|$, and each V_i contains $k+1$ variables: the k user-selected variables U along with exactly one additional (non-selected) variable $v_i \in D - U$, such that $V_i = U \cup \{v_i\}$. For example, if a user selects {`horsepower`}, Compass may return the field sets U = {`horsepower`}, V_1 = {`horsepower, cylinder`}, V_2 = {`horsepower, year`}, and so on.

Compass recommends all non-selected variables (all $v_i \in D - U$) by default, but analysts can interactively exclude fields from the suggestions to focus on a particular field set of interest (C2).

The generated field sets are returned in a sorted order. The user's selected field set is always ranked first, as it is the most relevant to the user's specified intention. The remaining field sets $[V_1, V_2, \ldots, V_n]$ are ordered by the type and name of the recommended variable v_i, consistent with the display order in Voyager's schema panel. This approach provides a predictable and consistent ordering, which works well for the common case of datasets with a sufficiently bounded number of variables to allow users to scroll through the recommendations. For datasets with a large number of variables, we plan to extend Compass to support multiple

relevancy rankings based on statistical measures (*e.g.,* [121]) and allow analysts to select measures that fit their interests.

5.5.2 Applying Data Transformations

For each suggested variable set $V \in \Sigma$ from the first phase, Compass enumerates applicable transformations for each field $v \in V$ to recommend an ordered set of data tables Γ.

Compass produces both *raw tables* without aggregation to provide details and *aggregate tables* to provide summaries. For raw tables, each variable is untransformed by default, but users can perform binning if desired. For aggregate tables, variables either serve as *measures* (values amenable to aggregation) or *dimensions* (values to group by). By default, each quantitative field is treated as a measure, while each ordinal, nominal, or temporal field is treated as a dimension.

Compass averages (`mean`) quantitative measures by default; users can choose to apply other aggregation functions such as `sum`, `min`, `max`. Averages may be affected by outliers or mix effects [25], but are also more likely to be generally useful. In our experience, defaulting to sums results in plots that are not always meaningful and skewed when the number of records varies across dimensions. Users can also choose to treat quantitative variables as dimensions by either using the untransformed values or binning. Compass determines the largest units of time within the extents of temporal fields. For example, if a variable spans within one year, `MONTH` is applied. Ordinal variables are untransformed by default.

Derived tables are first ordered by the rank of its corresponding variable set. (Tables derived from U come before tables from V_1, which in turn come before tables from V_2, and so on.) For tables from the same variable set, Compass then orders raw tables before aggregate tables to provide a consistent ordering (C4).

5.5.3 Designing Encodings

For each data table $T \in \Gamma$, Compass applies visualization design best practices drawn from by prior research [31, 51, 90, 91, 141] to generate and rank a set of encodings E_T (C3).

Generation. Compass first enumerates candidates for the encoding set E_T by composing permutations of data variables, visual encoding channels, and mark types. It first assigns each variable $v \in T$ to all permitted visual encoding channels (Table 5.1) to generate a set of mappings M_T. Then it generates each encoding candidate by combining each mapping $m \in M_T$ with each valid mark type.

Compass considers multiple criteria to determine whether a mark type is appropriate for a given mapping m. For a given mark type, some encoding channels are required, while some are disallowed (see Table 5.2). For example, Compass requires a mapping to have both `x` and `y` encodings if used with a `line` or `area` mark. Such constraints ensure the production of appropriate visualizations (here, a proper line or area chart). After determining the set of supported mark types, Compass assigns the mark type that best respects expressiveness criteria according to the rankings listed in Table 5.3, indexed by the data types of the `x` and/or `y` encodings.

In addition to Tables 5.1–5.3, Compass considers interactions among visual variables (*e.g.,* the perceptual separability of visual channels [141]) and avoids creating ineffective charts. It produces mappings that encode `color`, `size`, or `shape` only when the mappings also contain both `x` and `y`. In other words, Compass omits dot plots that use these encodings, as they would likely suffer from occlusion and visual clutter.

Ranking. Compass ranks the generated encodings using perceptual effectiveness metrics. Compass applies prior work by Cleveland [47] and Mackinlay [90] to rank the effectiveness of each visual channel based on a variable's data type (Table 5.1). Compass also considers the

Data Types	Encoding Channels
quantitative, temporal	x,y > size > color > text
ordinal	x,y > column, row > color > size
nominal	x,y > column, row > color > shape

Table 5.1: Permitted encoding channels for each data type in Compass, ordered by perceptual effectiveness rankings.

Mark Types	Required Channels	Supported Channels						
		X, Y	Column, Row	Color	Shape	Size	Detail	Text
point	x or y	✓	✓	✓	✓	✓	✓	
tick	x or y	✓	✓	✓				
bar	x or y	✓	✓	✓				
line, area	x and y	✓	✓	✓			✓	
text	text and (row or column)		✓	✓				✓

Table 5.2: Required and permitted encoding channels by mark type.

cardinality, or number of unique values, of a data variable. Encoding high cardinality variables with `color`, `shape`, `row`, or `column` can lead to poor color or shape discrimination or massive, sparse trellis plots. Moreover, the effectiveness of each visual channel is not measured in isolation. Since over-encoding can impede interpretation [141], Compass penalizes encodings that use multiple retinal encodings (*e.g.,* both `color` and `shape`, or both `color` and `size`).

Compass takes into account that the produced charts will be presented in a gallery, with the goal of promoting charts that are easier to read (C4). Vertical bar charts and histograms are preferred if their dimensions are binned quantitative or temporal fields. Otherwise, horizontal bar charts are preferred as their axis labels are easier to read. Horizontal line and area charts are favored over vertical ones. Compass also privileges encodings that use less

Data Types	Mark Types
Q	tick > point > text
(O **or** N) × (O **or** N)	point > text
Q × N	bar > point > text
Q × (T **or** O)	line > bar > point > text
Q × Q	point > text

Table 5.3: Permitted mark types based on the data types of the x and y channels. N, O, T, Q denote nominal, ordinal, temporal and quantitative types, respectively.

Positions	x, y
Facets	column, row
Level of detail	color (hue), shape, detail
Retinal measures	color (luminance), size

Table 5.4: Encoding channel groups used to perform clustering.

screen space, and hence are more easily browsed in the gallery. For example, colored scatter plots (Figure 5.4-A) are ranked higher than small multiple plots (Figure 5.5).

Compass maps all of the above features to scalar values and calculates a weighted sum to derive the effectiveness score s(e) for each encoding candidate e. We have manually tuned the current weights and scores for each feature through a series of refinements and tests. Automatic determination of these parameters remains as future work.

Clustering. To prevent exhaustive enumeration (C5), Compass groups encoding candidates that map the same variables to similar encoding channels listed in Table 5.4, and suggests only the most effective view in each group. This includes variants caused by swapping variables in the positional or facet encodings (producing transposed charts as depicted in Figure 5.9-D), or by selecting alternative retinal encodings (*e.g.*, `shape` instead of `color`). All

the suggested views are also sorted by the effectiveness score s. Therefore, for each $T \in \Gamma$, Compass produces an ordered set of visually different visualizations E_T, ranked by their perceptual effectiveness.

As a result, Compass recommends clusters of visualizations grouped by their corresponding data tables. To privilege data variation over design variation (C1), the main gallery presents the top ranked visualization for each table $T \in \Gamma$. When the user expands a view that shows data table T, the expanded gallery displays a set of visually different views E_T and provides an interface for refining presented views (Figure 5.4-C).

5.6 Evaluation: Voyager vs. PoleStar

We conducted a user study to contrast recommendation browsing with manual chart construction, focusing on exploratory analysis of previously unseen data. We compared Voyager with PoleStar, our own implementation of a visualization specification interface (Figure 5.10).

We hypothesized that Voyager would encourage breadth-oriented consideration of the data, leading to higher coverage of unique field combinations. Given its direct control over visual encodings, we expected PoleStar to be better for targeted question answering.

5.6.1 Study Design

Our study followed a 2 (visualization tool) $\times$ 2 (dataset) mixed design. Each participant conducted two exploratory analysis sessions, each with a different visualization tool and dataset. We counterbalanced the presentation order of tools and datasets across subjects.

Visualization Tools. Participants interacted with two visualization tools: Voyager and PoleStar, a manual specification tool. Rather than use an existing tool such as Tableau,

Figure 5.10: PoleStar, a visualization specification tool inspired by Tableau.

we implemented PoleStar (named in honor of Polaris [131]) to serve as a baseline interface, allowing us to control for external factors that might affect the study. Like Voyager, PoleStar models visualizations using Vega-Lite. In fact, any visualization suggested by Voyager can also be constructed in PoleStar, ensuring comparable expressivity. PoleStar also features similar UI elements, including field capsules, bookmarks, and an undo mechanism.

Figure 5.10 illustrates PoleStar's interface. The left-hand panel presents the data schema, listing all fields in the dataset. Next to the data schema are the encoding shelves, which represent each encoding channel supported by Vega-Lite. Users can drag and drop a field onto a shelf to establish a visual encoding. Users can also modify properties of the data (*e.g.*, data types, data transformations) or the visual encoding field (*e.g.*, color palette or sort order) via

popup menus. The mark type can be changed via a drop-down menu. Upon user interaction, PoleStar generates a new Vega-Lite specification and immediately updates the display.

Datasets. We provided two datasets for participants to explore. One is a dataset of motion pictures ("movies") comprising title, director, genre, sales figures, and ratings from IMDB and Rotten Tomatoes. The table has 3,201 records and 15 fields (7 nominal, 1 temporal, 8 quantitative). The other dataset is a redacted version of FAA wildlife airplane strike records ("birdstrikes"). The table has 10,000 records and 14 fields (8 nominal, 1 geographic, 1 temporal, 4 quantitative). We removed some fields from the birdstrikes data to enforce parity among datasets. We chose these datasets because they are of real-world interest, are of similar complexity, and concern phenomena accessible to a general audience.

Participants. We recruited 16 participants (6 female, 10 male), all students (14 graduate, 2 undergraduate) with prior data analysis experience. All participants had used Excel. Among other tools, 9 had used Tableau, 13 had used Python/matplotlib and 9 had used R/ggplot. No subject had analyzed the study datasets before, nor had they used Voyager or PoleStar (though many found PoleStar familiar due to its similarity to Tableau). Each study session lasted approximately 2 hours. We compensated participants with a $15 gift certificate.

Study Protocol. Each analysis session began with a 10-minute tutorial, using a dataset distinct from those used for actual analysis. We then briefly introduced subjects to the test dataset. We asked participants to explore the data, and specifically to "get a comprehensive sense of what the dataset contains and use the bookmark features to collect interesting patterns, trends or other insights worth sharing with colleagues." To encourage participants to take the analysis task seriously, we asked them to verbally summarize their findings after each session using the visualizations they bookmarked. During the session, participants verbalized their thought process in a think-aloud protocol. We did not ask them to formulate any questions before the session, as doing so might bias them toward premature fixation on

those questions. We gave subjects 30 minutes to explore the dataset. Subjects were allowed to end the session early if they were satisfied with their exploration.

All sessions were held in a lab setting, using Google Chrome on a Macbook Pro with a 15-inch retina display set at 2,880 by 1,980 pixels. After completing two analysis sessions, participants completed an exit questionnaire and short interview in which we reviewed subjects' choice of bookmarks as an elicitation prompt.

Collected Data. An experimenter (either the first or second author) observed each analysis session and took notes. Audio was recorded to capture subjects' verbalizations for later review. Each visualization tool recorded interaction logs, capturing all input device and application events. Finally, we collected data from the exit survey and interview, including Likert scale ratings and participant quotes.

5.6.2 Analysis & Results

We now present a selected subset of the study results, focusing on data field coverage, bookmarking activity, user survey responses, and qualitative feedback. To perform hypothesis testing over user performance data, we fit linear mixed-effects models [26]. We include visualization tool and session order as fixed effects, and dataset and participant as random effects. These models allow us to estimate the effect of visualization tool while taking into account variance due to both the choice of dataset and individual performance. We include an intercept term for each random effect (representing per-dataset and per-participant bias), and additionally include a per-participant slope term for visualization tool (representing varying sensitivities to the tool used). Following common practice, we assess significance using likelihood-ratio tests that compare a full model to a reduced model in which the fixed effect in question has been removed.

Voyager Promotes Increased Data Field Coverage

To assess the degree to which Voyager promotes broader data exploration, we analyze the number of unique field sets (ignoring data transformations and visual encodings) that users are exposed to. While users may view a large number of visualizations with either tool, these might be minor encoding variations of a data subset. Focusing on unique field sets provides a measure of overall dataset coverage.

While Voyager automatically displays a number of visualizations, this does not ensure that participants are attending to each of these views. Though we lack eye-tracking data, prior work indicates that the mouse cursor is often a valuable proxy [59, 72]. As a result, we analyze both the number of field sets shown on the screen and the number of field sets a user interacts with. We include interactions such as bookmarking, view expansion, and mouse-hover of a half-second or more (the same duration required to activate view scrolling). Analyzing interactions provides a conservative estimate, as viewers may examine views without manipulating them. For PoleStar, in both cases we simply include all visualizations constructed by the user.

We find significant effects of visualization tool in terms of both the number of unique field sets shown ($\chi^2(1, N = 32) = 38.056, p < 0.001$) and interacted with ($\chi^2(1, N = 32) = 19.968, p < 0.001$). With Voyager, subjects were on average exposed to 69.0 additional field sets (over a baseline of 30.6) and interacted with 13.4 more field sets (over a baseline of 27.2). In other words, participants were exposed to over 3 times more field sets and interacted with 1.5 times more when using Voyager.

In the case of interaction, we also find an effect due to the presentation order of the tools ($\chi^2(1, N = 32) = 5.811, p < 0.05$). Subjects engaged with an average of 6.8 more field sets (over the 27.2 baseline) in their second session.

Bookmark Rate Unaffected by Visualization Tool

We next analyze the effect of visualization tool on the number of bookmarked views. Here we find no effect due to tool ($\chi^2(1, N = 32) = 0.060, p = 0.807$), suggesting that both tools enable users to uncover interesting views at a similar rate. We do observe a significant effect due to the presentation order of the tools ($\chi^2(1, N = 32) = 9.306, p < 0.01$). On average, participants bookmarked 2.8 additional views (over a baseline of 9.7 per session) during their second session. This suggests that participants learned to perform the task better in the latter session.

Most Bookmarks in Voyager include Added Fields

Of the 179 total visualizations bookmarked in Voyager, 124 (69%) include a data field automatically added by the recommendation engine. Drilling down, such views constituted the majority of bookmarks for 12/16 (75%) subjects. This result suggests that the recommendation engine played a useful role in surfacing visualizations of interest.

User Tool Preferences Depend on Task

In the exit survey we asked subjects to reflect on their experiences with both tools. When asked to rate their confidence in the comprehensiveness of their analysis on a 7-point scale, subjects responded similarly for both tools (Voyager: $\mu = 4.88, \sigma = 1.36$; PoleStar: $\mu = 4.56, \sigma = 1.63$; $W = 136.5, p = 0.754$). Subjects rated both tools comparably with respect to ease of use (Voyager: $\mu = 5.50, \sigma = 1.41$; PoleStar: $\mu = 5.69, \sigma = 0.95$; $W = 126, p = 0.952$).

Participants indicated which tool they would prefer for the tasks of exploration vs. targeted analysis. Subjects roundly preferred Voyager for exploration (15/16, 94%) and PoleStar for question answering (15/16, 94%) – a significant difference ($\chi^2(1) = 21.125, p < 0.001$).

Finally, we asked subjects to rate various aspects of Voyager. All but one (15/16, 94%) rated Voyager's recommendations as "Helpful" or "Very Helpful". When asked if Voyager's inclusion of additional (non-selected) fields was helpful, 14/16 (88%) responded "Helpful" or "Very Helpful", with 2 responding "Neutral". We also asked participants to agree or disagree with the statement "The recommendations made by Voyager need improvement." Here, 8/16 subjects (50%) agreed with the statement, 5/16 (31%) were neutral and 3/16 (19%) disagreed. This last result surprised us, as we expected all subjects would request refined relevance rankings. In aggregate, these results suggest that though there remains room for improvement, the current Voyager system already provides a valuable adjunct to exploratory analysis.

Participant Feedback: Balancing Breadth & Depth

Participants' comments reinforce the quantitative results. Subjects appreciated Voyager's support for broad-based exploration. One said that *"Voyager gave me a lot of options I wouldn't have thought about on my own, it encouraged me to look more deeply at data, even data I didn't know a lot about".* Another *"found Voyager substantially more helpful in helping me learn and understand the dataset,"* while a third felt Voyager *"prompted me to explore new questions in a way that didn't derail me from answering follow-up questions."*

Subjects also reflected on the complementary nature of Voyager and PoleStar for the tasks of breadth- vs. depth-oriented exploration. One user noted that *"with Voyager, I felt like I was scanning the generated visualizations for trends, while with PoleStar, I had to think first about what questions I wanted to answer, then make the visualizations for them."* Another wrote that Voyager *"is really good for exploration but cumbersome for specific tasks."* All but one subject wished to use a hybrid of both tools in the future. For example, one participant said that *"if I have to just get an overview of the data I would use Voyager to generate visualizations, and then dive in deep using PoleStar,"* while another envisioned that *"I would start with Voyager but want to go and switch*

to PoleStar to dive into my question. Once that question was answered, I would like to switch back to Voyager."

5.7 Conclusion

In this chapter, we presented *Voyager*, a mixed-initiative system to facilitate breadth-oriented exploration in the early stages of data analysis. Voyager also contributes a visualization recommender system (*Compass*) to power a novel browsing interface that exchanges manual chart specification for interactive browsing of suggested views.

In a user study comparing Voyager with a visualization tool modeled after Tableau (*PoleStar*), we find that Voyager promotes broader exploration, leading to significantly greater coverage of unique variable combinations. The vast majority of participants (15/16) expressed a preference for using Voyager in future exploration tasks. Meanwhile, most of the participants (15/16) still prefer PoleStar for future question answering tasks.

Overall, this study indicated the value of chart recommendation for open exploration, but also called for a unified tool that supports both manual authoring and recommendation browsing. From the study results, we view Voyager as a first step towards improved systems that balance automation and manual specification. We consider Voyager's success for promoting broader exploration encouraging since PoleStar is based on a popular interaction model backed by over a decade of research and industrial use, whereas Voyager is relatively new and untested. A clear next step is to better integrate manual and automated chart authoring to faciliate both breadth-oriented and depth-oriented exploration. How might Voyager and PoleStar be most fruitfully combined? We further explore this question in Chapters 6-7.

6 CompassQL: Visualization Query Language & Recommender Engine

The user study in Chapter 5 shows the complementary benefits of recommendation browsing and manual chart specification for data exploration. Browsing recommended charts is less tedious and facilitates serendipitous discovery. Meanwhile, manual chart authoring gives user control to create arbitrary charts. This result motivates us to explore the design of interactive systems that blend both interaction models.

These two interaction models require different kinds of user inputs. Manual chart authoring tools [131, 144] typically demand users to provide *complete* specification of data and visual encodings. On the other hand, recommendation-powered chart browsers like Voyager only need *partial* specification. For example, Voyager lets users optionally select data fields and transformation functions to steer the recommendations. The system then automatically determines appropriate visual encodings for them. To combine these two interaction methods in a unified system, we need a shared model for representing both kinds of inputs.

To address this challenge, our approach is to design a language that can query over the space of visualization designs by blending a partial (incomplete) chart specification with directives for describing recommendation methods. We instantiate this approach in the CompassQL

query language, a generalization of the Vega-Lite visualization grammar (Chapter 4) to describe a chart recommendation query.

In this chapter, we first discuss typical chart recommendation process, including phases of enumerating, choosing, and ranking candidate visualizations. We then present the design of the CompassQL query language, which can express variations of the chart recommendation process. To process CompassQL queries, we introduce the CompassQL recommender engine, a derivation of the Compass engine in Chapter 5. Finally, we demonstrate the expressivity of CompassQL by showing how CompassQL queries can describe a variety of existing visualization recommendation systems.

Later in Chapter 7, we also show how CompassQL enables Voyager 2 to blend specification and recommendation in a single tool.

6.1 General Pattern for Visualization Recommendation Process

In Chapter 3, we review prior work on visualization recommender systems. These systems typically follow a similar process. First, a recommender system searches for a set of visualizations that match the user's intent. To perform this search, the system enumerates different data query and/or visual encoding parameters to produce specifications that satisfy a set of constraints. Given some satisfiable candidates, the system then orders recommendations based on a utility function. To avoid redundant recommendations, the system may group similar candidates and select a top-ranked representative from each group.

Enumeration. A system may enumerate data query parameters including selected fields, filters, aggregation, sorting, and field transformations such as binning. It may also enumerate

visual encoding parameters including encoding mappings between data fields and visual channels such as x-position, y-position, or color.

To qualify for recommendation, a candidate visualization must satisfy two types of constraints. First, it should respect any *data query or encoding constraints* based on the user's intent. Second, it must satisfy *expressiveness constraints* [90]. For example, a visualization that encodes a quantitative field with the shape of symbols is misleading since different shapes are unordered and therefore cannot convey the magnitude of quantitative values. Visualizations that use misleading encodings can be omitted from consideration.

Ranking. With a set of qualified candidates, the system then suggests the top item or produces an ordered list of recommendations based on various ranking functions.

Variations of encodings are typically ranked based on perceptual effectiveness metrics, applying prior works [47, 90] that study the effectiveness of different visual encoding channels based on the encoded data types.

To rank different data queries, a recommender system might use a naive order based on the data schema like our Voyager system in Chapter 5. This simple approach can work for datasets with small numbers of fields, but does not scale if there are many fields. With more fields, the system might take a data-driven approach to compute statistics that best suit a task. For example, if the user is interested in correlation between two fields, the system might rank visualizations based on Pearson's correlation coefficient [121]. If the user looks for anomalies in the data, the system might instead use the number of outliers or other metrics that measure deviation, normality, or uniformity to rank the recommendations [81,121,138].

For systems that recommend both visual encodings and data queries, designing a holistic ranking metric that takes both data query and encoding parameters into account can be complicated. Existing tools apply separate encoding-based or data-based metrics to rank the recommendations. The Small Multiples, Large Singles system [137] varies only one of either

the data query or the encoding parameter at a time, and thus can apply either type of metric directly. Our Voyager system groups visualizations backed by the same data query, applies an encoding-based metric to choose group representatives, and orders them using a data-based metric.

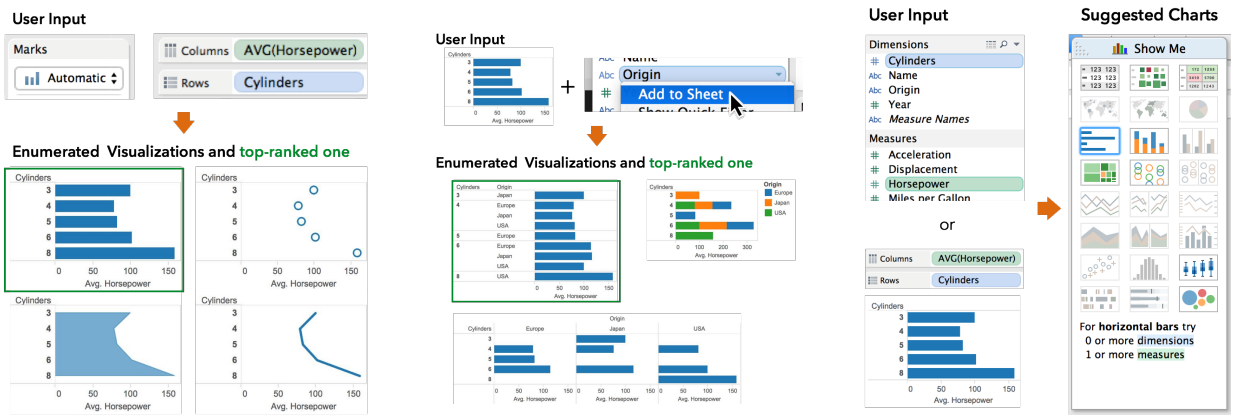
Reducing Redundancy. Many candidate visualizations may be similar and thus redundant, increasing the number of charts the user has to consider without providing additional value. For example, it is often unnecessary to suggest both horizontal and vertical bar charts of the same data fields. A recommender system might group similar charts to reduce redundancy and encourage diversity. For example, the Show Me Alternatives feature in Tableau [91] suggests only one instance for each basic chart type, as shown in Figure 6.1–C. To support broader exploration, Voyager groups visualizations with the same data query and only presents the most perceptually effective item from each group in its main view (Figure 5.1).

6.2 The CompassQL Visualization Query Language

To describe all phases of the visualization recommendation process identified in the previous section, we present the CompassQL visualization query language.

A CompassQL query (*e.g.*, in Figure 6.1) consists of (1) a partial chart specification and enumeration constraints, and (2) methods for choosing, ordering, and grouping suggestions. With CompassQL, a recommender system may set some query parameters as a part of a template, with other parameters interactively specified by users. For illustration, in subsequent figures we highlight user provided parameters in red.

Partial Specification. A specification in CompassQL (`spec`) has a similar structure to a Vega-Lite unit specification (Chapter 4). However, a CompassQL query specification allows replacing concrete values with *wildcards* (denoted with bold, capital letters in the section), indicating



AUTOMATIC MARKS

Partial Specification

Mark	**M**
Encoding	x: Mean(Horsepower) y: Cylinders

Recommendation Method

Choose by	Effectiveness(s,D)

ADD TO SHEET

Partial Specification

Mark	bar
Encoding	x: Mean(Horsepower) y: Cylinders **C**: Origin

Recommendation Method

Choose by	Effectiveness(s,D)

SHOW ME ALTERNATIVES

Partial Specification

Mark	**M**
Encoding	**C1**: Mean(Horsepower) **C2**: Cylinders

Recommendation Method

Group by	ChartType(s)
Choose by	Effectiveness(s,D)
Order by	Effectiveness(s,D)

Figure 6.1: CompassQL can be used to express existing Show Me features in Tableau [91], which rank effectiveness of each specification s for dataset D. (A) *Automatic Marks* determines the most effective mark type for the specified data query and encoding mappings. (B) *Add To Sheet* recommends the most effective encoding mapping for a new field added to an existing visualization. (C) *Show Me Alternatives* suggests chart types for provided data fields. The interface highlights the most effective chart type with blue border.

that certain properties should be determined by the query engine. For example, setting the `mark` property in Figure 6.1-A to a wildcard (**M**) means that the system should enumerate all possible mark types (*e.g.*, `bar`, `line`, `area`, `point`). Wildcards can also include user-defined constraints. For instance, the aggregate function **A** in Figure 6.6 is constrained to either `-` (no aggregate) or `mean`.

Besides user-defined constraints, CompassQL implicitly applies a set of *expressiveness* constraints (listed in Appendix A). Derived from Voyager's Compass engine in Chapter 5, these constraints concern both perceptual expressiveness [90] and the expressiveness of transformation functions. For example, CompassQL excludes charts that use *shape* to encode a quantitative field (as *shape* does not convey magnitude), and avoids inapplicable functions (*e.g.,* by not applying *time units* to non-temporal fields).

Choosing and Ordering. A query might specify the `choose by` property to define how to choose the top recommendation, or set the `order by` property to produce a ranked list recommendations in decreasing order of scores.

Both the `choose by` and `order by` properties refer to ranking methods. These methods are either provided natively by the query engine (such as the encoding `effectiveness` that we derive from the prior Compass engine in Chapter 5) or defined by developers as user-defined functions.

These ranking methods can take a visualization specification s, the data relation D, and values of wildcards as input. Some ranking methods such as encoding `effectiveness` may depend only on the specification and simple statistical properties such as cardinality. However, data-driven metrics such as mutual information and deviation require more expensive data computation. A ranking method might also invoke an external module such as a learning model that improves over time. For example, Figure 6.1-A shows a query for Show Me Automatic Marks [91], which attempts to suggest the most effective mark; the query `choose` a specification s that maximizes the `effectiveness` score. In Figure 6.2-A, the query sets `order by` property to a user-selected ranking R_1 of the enumerated field **F** to produce an ordered list of histograms.

Grouping. To reduce redundancy, a query can include a `group by` clause, providing a key function to induce groupings. When `group by` is provided, the `choose by` property defines how the system chooses a top representative for each group; meanwhile, the `order by`

property specifies how the system orders the representative from each group in the list of recommendations. For example, in Figure 6.6, Voyager groups visualizations by matching data queries, and chooses the most perceptually effective visualization to represent a group; the chosen visualizations are then ordered by query simplicity: raw plots are shown before aggregate plots. Similar to the ranking methods, system developers may provide user-defined key functions to the query engine.

6.3 The CompassQL Visualization Query Engine

We now present the CompassQL query engine, which is a derivation of the Compass engine in Chapter 5 to process CompassQL queries. Given a CompassQL query, the engine generates recommended charts in the form of Vega-Lite specifications. Following the common chart recommendation process in Section 6.1, the engine produces recommendations in three phases: *enumerate*, *group*, and *rank*.

Enumerate. The engine first *enumerates* possible candidate views that satisfy all constraints using a backtracking algorithm [111]. In this process, the engine incrementally replaces each wildcard in the input partial specification with all possible values for that property. For each value assigned to replace a wildcard, the engine creates a *partial candidate* (a partial specification with one less wildcard) and tests it against all constraints relevant to the assigned property. The engine continues replacing other wildcards in a partial candidate only if it satisfies all constraints; the engine "backtracks" if the candidate fails any constraint. Whenever the engine successfully replaces all wildcards with a set of values, it includes the candidate (now a complete Vega-Lite specification) in the answer set. The engine continues this search process until the answer set includes all valid candidates.

The enumeration order of properties in a specification can affect the performance of the backtracking algorithm (also known as the *variable ordering* problem [111]). In response, we

apply the *fail-first* heuristic [61]: we enumerate properties with the highest likelihood of causing violations first.

We first enumerate values for each *encoding* property, as they have a number of constraints independent of other parts of the specification. For each potential encoding, we start by considering the *data-type*, as it has relationships with most other encoding properties: a *field* must match the data type, most aggregate *functions* do not apply to nominal or ordinal fields, *bin* is applicable only to quantitative fields, and the expressiveness criteria of a *channel* depends on the data type (*e.g., shape* cannot encode quantities). We then enumerate data *field*, transformation *function* and *channel* values in turn, as the expressiveness of a channel depends on the cardinality of a field and whether a continuous field is binned. Finally, we enumerate *mark* types, as mark-related constraints may depend on the encoding properties that have been specified.

Group. For each candidate in the enumerated answer set, the engine then applies the *group-by* key function and groups items based on the resulting key values.

Rank. Using the specified ranking functions, the engine calculates ranking scores for all candidates in the answer set. Using these scores, the engine ranks the candidates within each group; the top-scoring candidate is used as the exemplar for the group. The engine then orders the group exemplars to produce the ranked list of recommendations.

6.4 CompassQL Queries for Existing Visualization Recommender Systems

To demonstrate the expressivity of CompassQL, we now illustrates how we can use CompassQL queries to express a variety of prior chart recommendation approaches including related work in Chapter 3 and the Voyager system in Chapter 5.

6.4.1 Encoding Recommendations

Encoding recommenders generate variations of visual encodings for a fixed, user-provided data query. Their recommendation queries have wildcards for mark type or visual channels in the encoding mappings. All of them rank outputs based on their encoding effectiveness.

Show Me [91] is a set of features that provides automation to facilitate the creation of visualizations in Tableau. Figure 6.1-A shows a query for *Automatic Marks*. Since the `mark` property is specified as enumerable, the system generates multiple candidate visualizations by varying the mark type and recommends the most effective presentation. In Figure 6.1-B, the user selects a field (`Origin`) to add it to the view with *Add To Sheet*. The system enumerates and ranks alternative mappings between the selected field and available encoding channels. Figure 6.1-C shows a query for *Show Me Alternatives*, which recommends alternative chart types for selected fields or fields of the current chart, if users do not select any fields. All channels in the encoding mappings and *marks* are specified as enumerable. The recommendations are shown as a list of compatible chart types, with the top-ranked type highlighted. If the user selects a chart type that has multiple compatible encoding mappings, Show Me recommends the top-ranked encoding.

APT [90] and **Spotfire Recommendations [3]** suggest encodings for selected fields akin to Show Me Alternatives (Figure 6.1-C) and thus have similar partial specifications. However, APT only recommends the top result (`choose by` = Effectiveness(s, D)). Meanwhile, Spotfire appears to suggest variations of the same chart type with different encoding mappings and thus does not `group by` chart types.

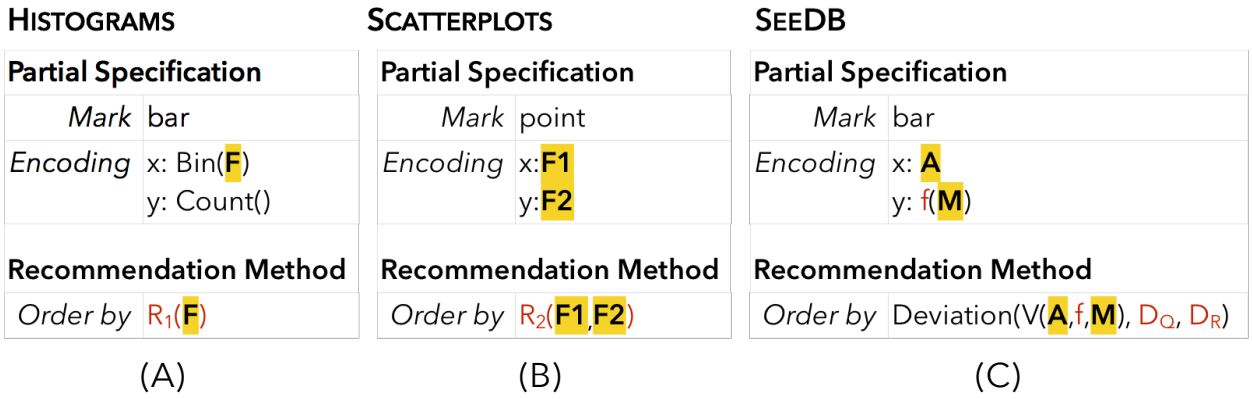


Figure 6.2: Data query recommendations. (A–B) Rank-by-Feature Framework [121] queries for ranking histograms and scatterplots by the selected metrics R_1 and R_2. (C) a query for ranking bar charts of aggregate views V with varying dimension A and measure M by the deviation between select target and reference data subsets (D_Q, D_R) akin to SeeDB [138].

6.4.2 Data Query Recommendations

CompassQL queries that recommend the data to view fix the visual encoding templates and only have wildcards for data query parameters such as the data fields.

The **Rank-by-Feature Framework [121]** ranks histograms and scatterplots based on selected metrics. In the queries shown in Figure 6.2 (A–B), the metrics R_1 and R_2 are functions of the enumerated fields (**F** for histograms; **F1** and **F2** for scatterplots). The framework provides many metrics such as distribution normality and distribution uniformity for histograms as well as correlation and the number of potential outliers for scatterplots. Other works such as **Scagnostics [147]** propose a set of cognostics [134], or metrics that measure the relative interestingness of different displays, for ranking scatterplots as well.

SeeDB [138] suggests bar charts of aggregate views with the highest deviation between the user provided target and reference data subsets (D_Q and D_R). Figure 6.2-C shows a

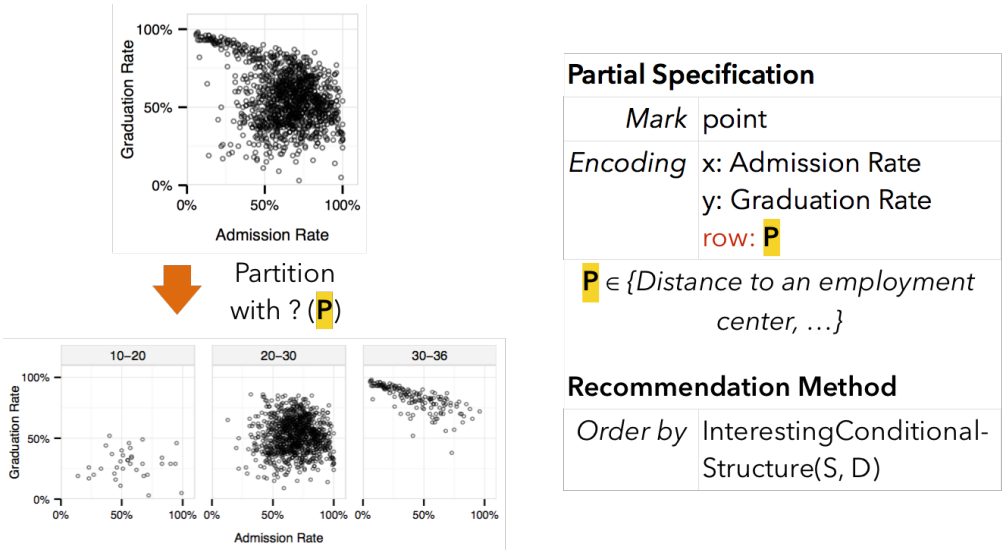


Figure 6.3: Automatic Selection of Partitioning Fields for Small Multiple Displays [23] can be expressed as a CompassQL query that enumerates partition fields and ranks them based on a randomized, non-parametric permutation test that determines interesting conditional structure in the data.

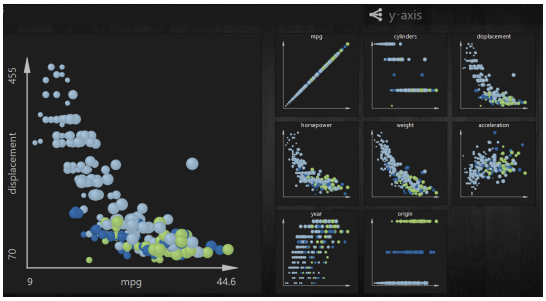
recommendation query akin to this suggestion[1]. The system ranks each aggregate view V that computes a selected aggregate function f of field **M** grouped by field **A** based on the deviation between the data subsets, which is defined as the distance between their probability distributions:

$$\text{Deviation}(V, D_Q, D_R) := S(P[V(D_Q)], P[V(D_R)])$$

where S is a distance function (*e.g.*, earth mover's distance) and P[V(D)] is a probability distribution of V given a data subset D.

Automatic Selection of Partitioning Variables for Small Multiple Displays [23] is a method to rank partitioning fields that reveal interesting pattern in the data with a randomized, non-parametric permutation test and cognostics [134]. The query shown in Figure 6.3 varies the

[1] SeeDB presents recommended views as grouped bar charts that juxtapose each dimension's value of both data subsets. Our formulation enables an implementation of an equivalent display that shows small multiple bar charts of the recommended view (each multiple represents each data subset).



Partial Specification

Mark	point
Encoding	x: Miles_per_Gallon y: **F**

Recommendation Method

Order by	SchemaIndex(**F**)

Figure 6.4: The Small Multiples Large Singles system [137] when a user chooses to enumerate the field on y-axis, expressed as a CompassQL query.

partition field **P**, which is mapped to the `row` channel to create small multiples of scatterplots, and ranks each small multiple based on this method.

6.4.3 Hybrid Recommendations

Hybrid recommendation queries can have wildcards for both data query and encoding parameters.

Small Multiples, Large Singles [137] shows small multiple displays that are variants of a main display. The variants are produced by changing either a data query (filtering data, changing an axis) or a visual encoding parameter (changing size, color, or the parameters of a layout algorithm). For example, Figure 6.4 shows the interface and a CompassQL query with the field **F** on the y-axis varied.

VizDeck [104], as shown in Figure 6.5, displays a ranked list of 1D and 2D visualizations which the user can vote up or down. The system ranks the results using a combination of heuristics and a model of visualization quality that learns the relationship between summary statistics of the data (*e.g.*, entropy, coefficient of variation, kurtosis, and periodicity) and voting feedback collected by the interface.

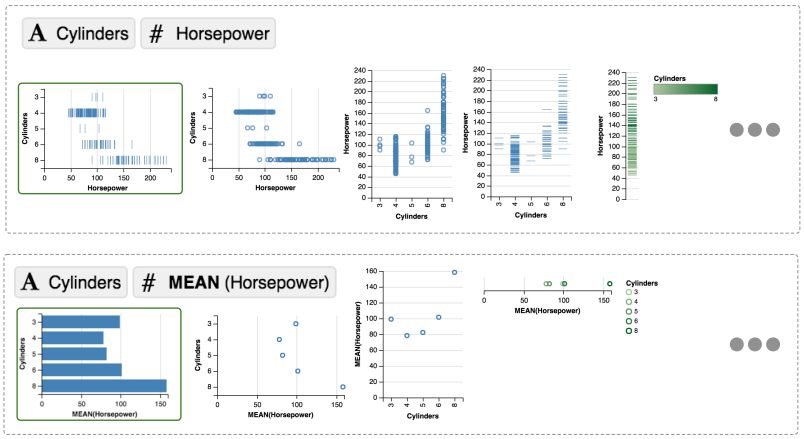
Partial Specification

Mark	**M**
Encoding	**C1**: **A1**(**F1**) **C2**: **A2**(**F2**) (Optional)
	A1, **A2** ∈ {-, *Mean*}

Recommendation Method

Order by	RankingModel(S, D, userVotes)

Figure 6.5: VizDeck [104] showing 1D and 2D visualizations ordered by a ranking model trained with user votes, expresses as a CompassQL query.



Partial Specification

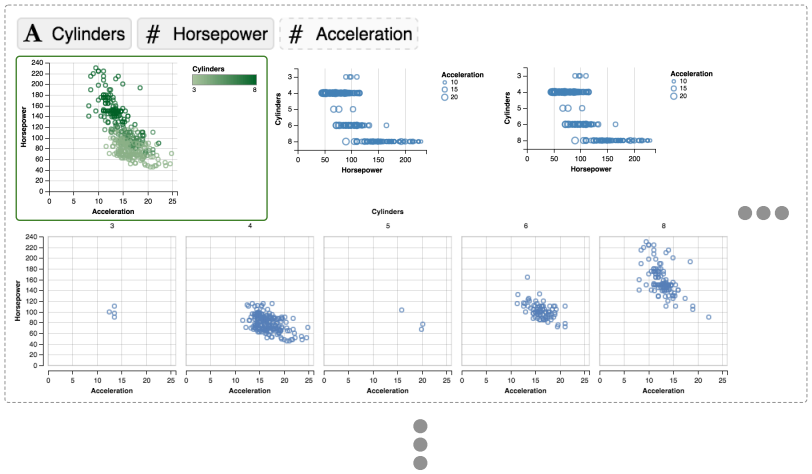
Mark	**M**
Encoding	**C1: A**(Horsepower) **C2**: Cylinders
	A ∈ {-, *Mean*}

Recommendation Method

Group by	DataQuery(s)
Choose by	Effectiveness(s,D)
Order by	QuerySimplicity(s)

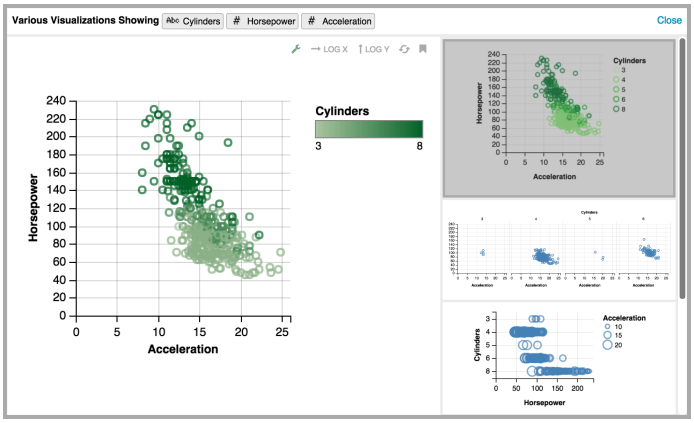
Figure 6.6: Enumerated charts (left) and its CompassQL query (right) for Voyager's *exact match* section in Figure 5.1.

Voyager (from Chapter 5) suggests visualizations based on selected fields, showing both variations of data queries and visual encodings. The *main view* (Figure 5.1–A) has two sections that display visualizations for different data queries to encourage exploration of different fields. The *exact match* section lists visualizations of selected fields with varying aggregation levels. The *suggestion* section adds an extra field to promote further exploration. Figures 6.6 and 6.7 illustrate recommendation queries for the *exact match* and *suggestion* sections. Both queries group results by data query and suggest the top chart from each cluster. The



Partial Specification	
Mark	**M**
Encoding	**C1**: **A1**(Horsepower) **C2**: Cylinders **C3**: **A2**(**F**)
	A1, A2 ∈ {-, *Mean*}
Recommendation Method	
Group by	DataQuery(s)
Choose by	Effectiveness(s,D)
Order by	SchemaIndex(**F**), QuerySimplicity(s)

Figure 6.7: Enumerated charts (left) and its CompassQL query (right) for Voyager's *suggestion* section in Figure 5.1.



Partial Specification	
Mark	**M**
Encoding	**C1**: Horsepower **C2**: Cylinders **C3**: Acceleration
Recommendation Method	
Group by	SimilarEncodingKey(s)
Choose by	Effectiveness(s,D)
Order by	Effectiveness(s,D)

Figure 6.8: Voyager's *expanded View* (left) and its CompassQL query (right).

exact match section orders visualizations by data query simplicity, showing raw plots before aggregate plots. The *suggestion* section orders visualizations by schema indices of the added fields, then by the same query simplicity criteria. Users can browse different encodings of a particular data query in the *expanded view* (Figure 6.8). Its query is similar to encoding recommenders. It enumerates only mark and encoding mappings, and then ranks outputs by encoding effectiveness. To reduce redundancy, the *expanded view* also groups visualizations

with similar encodings and only suggests one visualization for each cluster. For example, it does not suggest another scatterplot that is a transpose of the scatterplot shown in Figure 6.8 (left).

6.5 Conclusion

In this chapter, we present the design of CompassQL visualization query language and engine. To define a chart recommendation query, CompassQL combines partial chart specification with directives for grouping redundant plots and choosing or ordering plots. Given a query, the CompassQL engine then enumerates, ranks, and groups recommendations to produce query results. We demonstrate the expressivity of CompassQL by showing CompassQL queries for a variety of existing visualization recommendation systems.

In Chapter 7, we will use CompassQL as an underlying representation for Voyager 2, an interactive system that blends chart specification and recommendation. Beyond this thesis, CompassQL was also used to generate training data for an analysis pipeline that reverse-engineers visual encodings from bitmap chart images [106]. A recent project called Draco [98] also builds on Vega-Lite and CompassQL to develop a chart recommender engine based on answer-set programming, making it easier to extend the constraints and learn to rank preferences.

7 Voyager 2: Augmenting Visual Analysis with Partial View Specifications

Exploratory data analysis is an iterative process that involves both open-ended exploration and targeted question answering analysis [64, 65, 135]. As this process is iterative by nature, an analyst's focus on open-ended exploration and targeted question answering can vary throughout an analysis. However, existing tools provide interaction models designed primarily for either focused question-answering or open-ended exploration.

Traditional visual analysis tools (*e.g.,* [131, 144, 146]) provide specification interfaces for creating an expressive range of visualizations, making it easy for analysts to answer a variety of questions. Yet providing complete view specifications can be tedious and require domain familiarity as well as design and analysis expertise, making it inconvenient to systematically coverage different aspects of the data.

In Chapter 5, we introduce the Voyager visualization browser, which facilitates broad exploration by suggesting data and views for analysts to browse. However, while Voyager allows users to steer the recommendations by selecting data fields, it still provides limited fine-grained control of the visualizations. As shown in the prior user study, the recommen-

dation browsing approach in Voyager can be insufficiently expressive for focused question answering.

Rather than treat analysis as a process with dichotomous "modes," analysts may be better served by tools that support smooth gradations between open-ended and more focused phases of analysis. Towards this goal, we present *Voyager 2*, a new mixed-initiative tool that blends manual and automatic chart specification in a unified system. Voyager 2 augments traditional visual analysis interfaces with two new *partial view specification* techniques. Using *wildcards*, analysts can precisely vary the properties of a specification to generate multiple charts in parallel, giving them control over sets of views aligned with their analysis goals. *Related views* automatically recommends charts based on the current user-specified *focus view*, promoting discovery of relevant data fields and alternative ways to summarize or encode the data.

In this chapter, we first motivate the design of Voyager 2 with a usage scenario and a set of design considerations. Both of which are revisions of the usage scenario and design considerations of the original Voyager system in Chapter 5. We then introduce the interface design of Voyager 2, which blends multiple interaction methods for both manual and automated chart specification. We also describe how the *CompassQL* visualization query language from Chapter 6 enables specifications and recommendations in Voyager 2.

We evaluate Voyager 2 in a controlled user study comparing it with PoleStar, a view specification tool modeled on Tableau. Analysis of usage logs and subject ratings finds that Voyager 2 leads to increased data field coverage and higher ratings for open-exploration tasks. Meanwhile, subjects rate Voyager 2 comparably with PoleStar for targeted question answering. Comparing these results with a prior comparative study of Voyager and PoleStar [150], we find that Voyager 2 improves upon these prior systems in terms of supporting *both* open-ended and focused analysis.

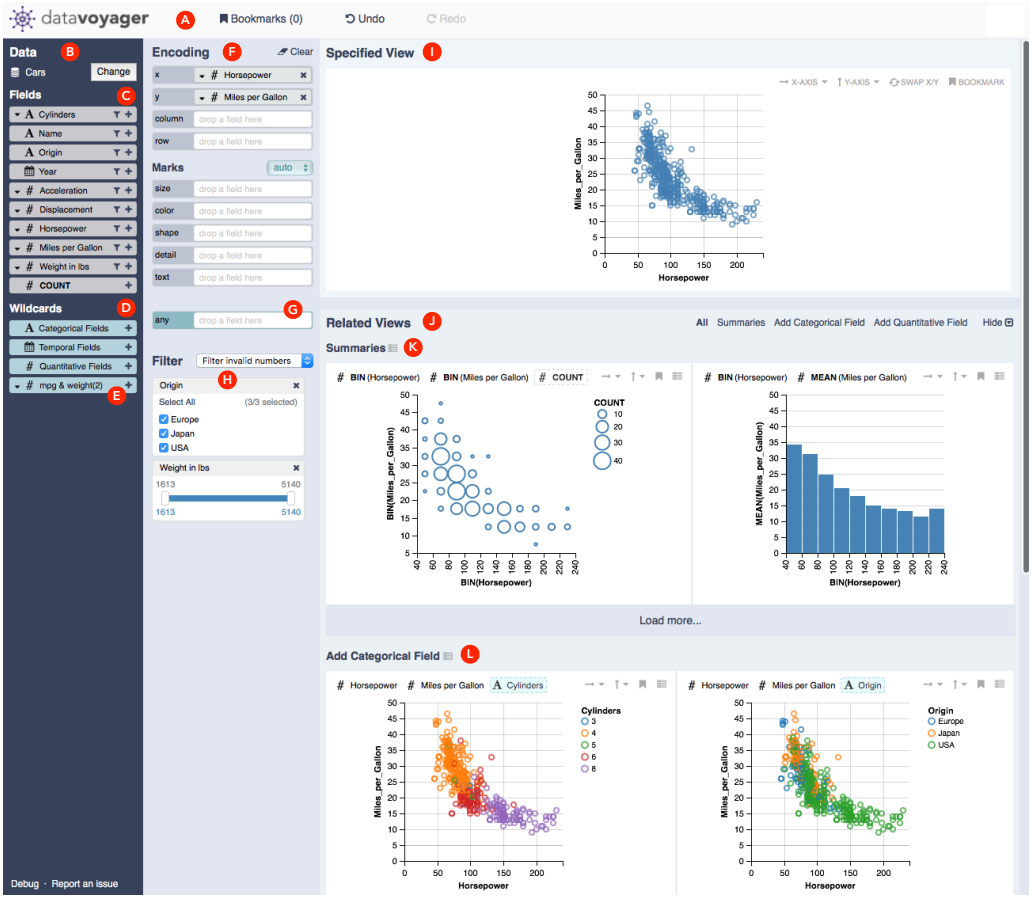


Figure 7.1: The Voyager 2 Interface. The top panel (A) provides bookmark gallery and undo commands. The *data* panel (B) contains the dataset name, data fields (C), and *wildcard fields* (D). Wildcard fields let users create multiple views in parallel by serving as "variables" over an enumerated set of fields. *Categorical*, *temporal*, and *quantitative field* wildcards are provided by default, though users can manually author *custom wildcards* containing desired fields (E). The *encoding* panel (F) contains shelves for mapping fields to visual channels via drag-and-drop, and a control for selecting mark type. A *wildcard shelf* (G) lets users add fields without selecting a specific channel, allowing the system to suggest appropriate encodings. The *filter* panel (H) shows dynamic query controls for filtering. The primary *focus view* (I) displays the currently specified chart. *Related views* (J) show recommended plots relevant to the focus view. Related *summaries* (K) suggest aggregate plots to summarize the data. *Field suggestions* (L) show the results of encoding one additional field within the focus view.

Figure 7.2: Upon loading a dataset, the *focus view* (A) is empty. The *related views* show *univariate summaries* (B) for all fields.

7.1 Usage Scenario

We start by describing how an analyst might use Voyager 2 to explore data. To aid comparison with the example usage scenario of the original Voyager system (Section 5.1), we use the same dataset of automobile statistics [110]. Upon loading the data, the analyst first wants to familiarize herself with the dataset. She attends to the univariate summaries in the *Related Views* panel (Figure 7.2). She sees that most, though not all, cars have an even number of

cylinders. She also notes multiple records containing the same model name. The majority of cars originate from the USA, though others are from Europe or Japan; all made in the years 1970–1982. The histogram of *acceleration* appears normally distributed, while histograms of other quantitative fields more closely resemble log-normal distributions.

After assessing each univariate summary, the analyst wishes to examine potential associations among quantitative fields. To specify a set of bivariate plots for each pair of fields, she drags two *quantitative field wildcards* (Figure 7.1-D) to the wildcard "any" shelves (Figure 7.1-G). In turn, Voyager 2 automatically chooses encodings by mapping each pair of quantitative fields to the *x* and *y* axes to produce a gallery of scatter plots (Figure 7.3). Scrutinizing these plots, she observes a roughly quadratic relationship between *horsepower* and *miles per gallon* (Figure 7.3-A). As she is interested in investigating this relationship further, she bookmarks the plot and adds a text note documenting her observation. She then clicks the *specify* button (▤) to make this plot the new *focus view* (Figure 7.1-I).

The first section of *Related Views* now shows summary plots (Figure 7.1-K), including a 2D histogram of *horsepower* and *miles per gallon*. The second section presents variants of the focus scatter plot with additional color-coded categorical fields (Figure 7.1-L). The analyst notices that the USA is the *origin* of cars with high *horsepower* and low *miles per gallon*. To dig deeper, she makes this plot the new focus view and examines its related summary plots (Figure 7.4-A). She confirms that the mean *horsepower* of American cars is higher than other regions, while the mean *miles per gallon* is lower. Below, she also sees a suggested trellis plot partitioned by *origin* as an alternative to color-coding (Figure 7.4-B).

At this point, the analyst wonders how *origin* affects other characteristics of the cars. She clears the encoding shelves and adds *origin* and a *quantitative field wildcard* to the shelves (Figure 7.5-A). To see both raw distributions and mean summaries, she also applies a *wildcard function* to the quantitative field wildcard (Figure 7.5-A, 7.5-C). She examines the resulting

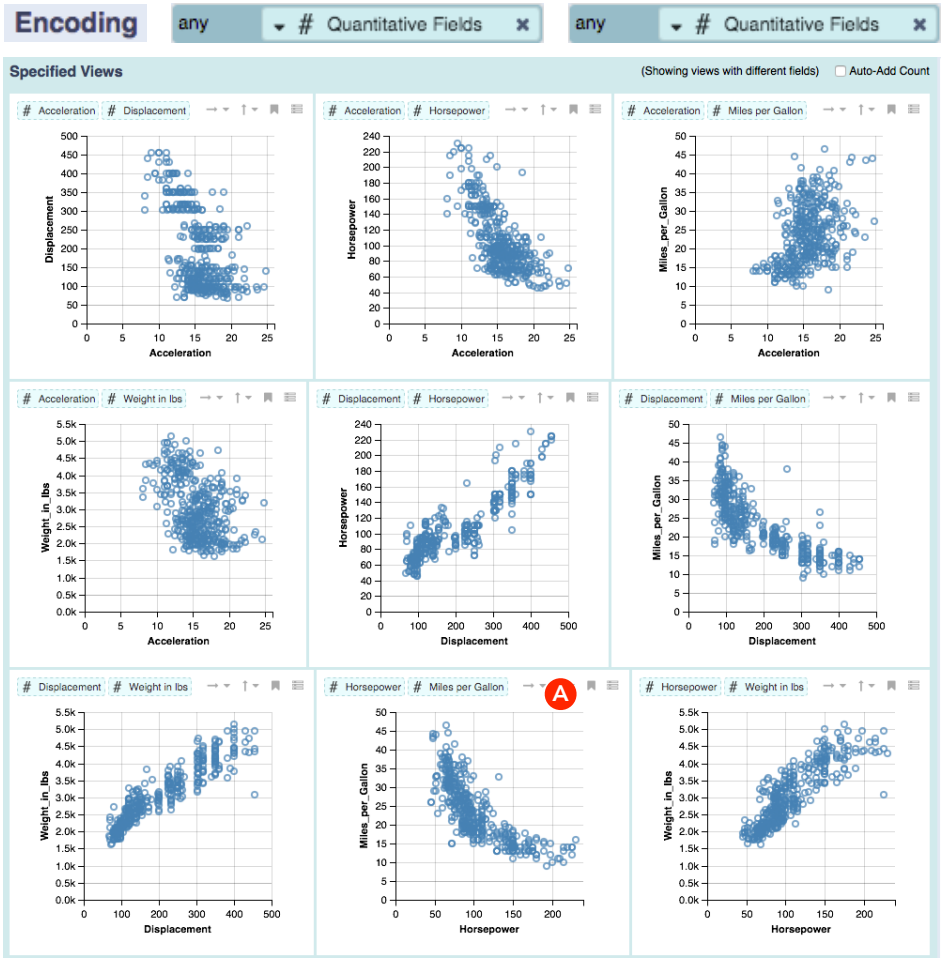


Figure 7.3: Dropping two *quantitative field wildcards* onto the *wildcard shelves*. Voyager 2 automatically chooses encodings, producing scatterplots that show bivariate relationships between all quantitative fields.

gallery of strip plots and aggregate bar plots in Figure 7.5-D to assess the relationships between *origin* and all quantitative fields at multiple levels of detail.

The analyst continues her analysis, exploring various aspects of the data. As she discovers insightful views, she bookmarks and annotates them with notes so that she can subsequently share her findings with colleagues.

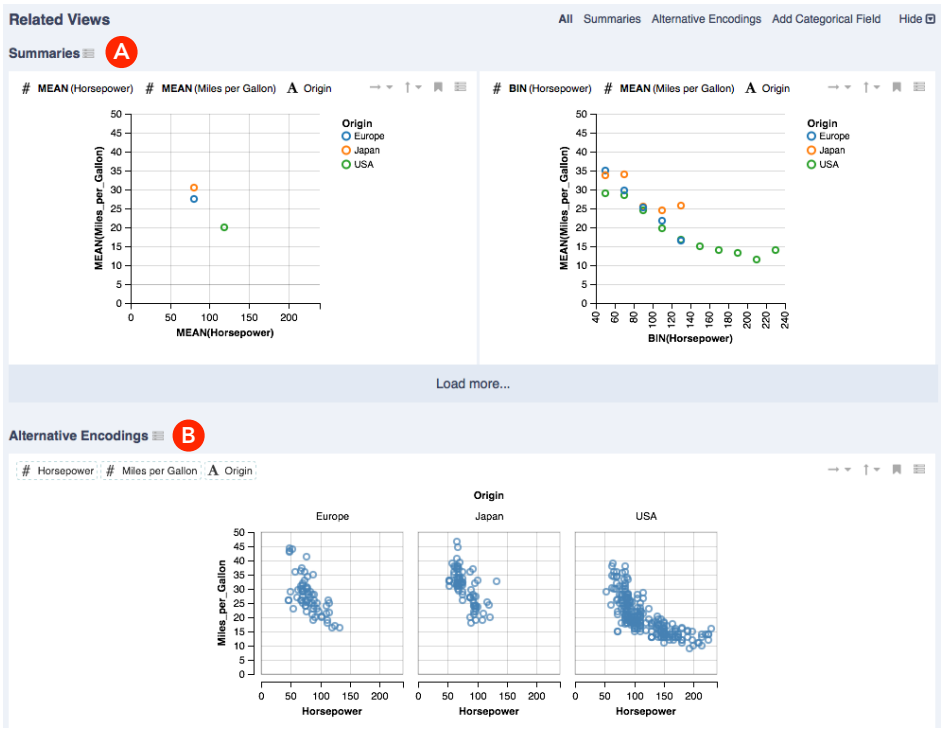


Figure 7.4: Setting the focus view to the colored scatterplot of *horsepower*, *miles per gallon*, and *origin* in Figure 7.1-L. The related views panel then displays *summaries* (A) and *alternative encodings* (B) of the focus view.

7.2 Design Considerations

In Section 5.2, we propose a set of considerations to guide the design of recommendations for the Voyager faceted chart browser. Here, we revisit those considerations (C1-C6) and propose extensions (C7, C8) in the context of a unified tool that blends manual and automatic chart specifications.

C1. Show data variation, not design variation. *Recommendations should prioritize views of different fields and transformations over different encodings of the same data.* Related views and wildcards in Voyager 2 continue to follow this approach, encouraging broader data exploration [96]. However, Voyager 2 also provides precise control over visual encodings.

Figure 7.5: Mapping a *quantitative field wildcard* to *x* and *origin* to *y* (A) produces a gallery of plots. A *wildcard function* enumerates no function (*none*) and *mean* (B–C), generating strip plots of raw values and bar charts of mean values (D). The ? in (A) denotes the wildcard function.

C2. Allow interactive steering to drive recommendations. *The system must provide controls for users to indicate their intent.* Voyager lets users select data fields and summary functions of interest, but does not permit encoding specification. Voyager 2 blends recommenders into a manual specification tool, granting more control to analysts. Users can create arbitrary encodings as in Tableau and browse *related views* based on the current focus, or query a set of views with *wildcards*.

C3. Use expressive and effective visual encodings. *Recommenders should apply perceptual design principles [31, 47, 90].* Voyager 2 continues to apply *expressiveness* criteria to exclude

misleading encodings and uses *effectiveness* metrics to rank suggestions as a part of the underlying CompassQL language.

C4. Promote reading of multiple charts in context. *Present related charts such that effort spent reading one chart can aid reading of the next.* Akin to Voyager, Voyager 2 aligns charts and makes their axis ranges consistent to ease comparison when possible. Voyager 2 also applies this consideration for suggesting summary views or views with additional fields by preserving the visual encodings of the focus chart. It also clusters suggestions into groups to provide local consistency.

C5. Prefer fine-tuning to exhaustive enumeration. *Rather than show all possible charts, include simple interactions to view chart variants.* Voyager 2 preserves lightweight interactions for sorting, scale transforms, and axis transposition of recommended charts. In addition, its unified specification interface enables fine-tuning of visual encoding mappings.

C6. Enable revisitation and follow-up analysis. *Provide bookmarking and export features to enable sharing and recall.* Voyager 2 provides similar support for undo, bookmarks and chart export. Bookmarks now include support for text notes.

For Voyager 2, we further extend these considerations:

C7. Use automation to extend user focus. *Ground suggestions in the current context of analysis.* While Voyager presents browseable recommendations, Voyager 2 instead augments manual specification. *Related views* elaborate on the user's focus view, while *wildcards* allow precise control over chart exploration. To transition from browsing to follow-on analysis, users can make any suggested or enumerated views the new focus, or interact with the shelves to modify the view.

C8. Avoid redundant suggestions. *Presenting many similar recommendations may overwhelm or distract users.* In addition to promoting data variation over design variation, Voyager 2 groups

suggestions into selectable categories, limits the default number of suggestions per category, and prunes the space of visual encodings to suggest distinct designs.

7.3 The Voyager 2 Interface Design

We now present the interface design of Voyager 2, which allows users to pivot among manual specification, *wildcard* specification, and browsing *related views* in a unified system.

7.3.1 Basic Interactions for Manual Chart Specification

Figure 7.1 shows the Voyager 2 interface. The top panel provides buttons for undo, redo, and opening a bookmark gallery (C6). The left column contains the *data* panel, which lists data fields and wildcards, ordered by data type and then by name. A *count* field is provided to aggregate the number of records.

The middle column contains the *encoding* and *filter* panels. The *encoding* panel (Figure 7.1-F) provides controls for specifying visual encodings and data field transformations (C2). To visualize data, analysts can drag-and-drop a data field onto an encoding channel shelf (*e.g.,* *x*, *y*, *color*). By default, the system encodes raw (unaggregated) values. Via a drop-down menu (Figure 7.5-B), analysts can select a transformation such as an aggregation, binning, or time unit function.

We also provide automatic encoding features akin to Tableau's *Add to Sheet* and *Automatic Mark* [91]. Analysts can double-click a field or click its *add to shelf* (+) button to add it to an automatically-selected encoding channel (C3). By default, the system automatically picks an appropriate mark type; analysts can override this choice using a drop-down menu. To enable dynamic query filters [124] as in Figure 7.1-H, a user can drag a field onto the *filter* panel or click the filter button (▼).

As encodings are modified, the *focus view* (Figure 7.1-I) and *related views* (Figure 7.1-J) panels in the rightmost column update accordingly. Each view contains toolbar buttons for modifying axes, sorting, and bookmarking (C5, C6). Users can hover over a chart element to reveal a tooltip describing the underlying data point. For large views that require scrolling, users can activate a local scroll bar by hovering for 500ms (to disentangle local and global scrolling), akin to Voyager (Chapter 5.

7.3.2 Specifying Views in Parallel with Wildcards

Wildcards let analysts specify multiple charts in parallel by authoring partial specifications (C2, C7). In response, Voyager 2 presents a *specified gallery*, showing charts that satisfy the wildcard constraints, as in Figures 7.3, 7.5, and 7.6. The specified gallery and other wildcard interfaces use teal background to distinguish them from other items. To avoid overwhelming users (C8), the system does not show *related views* when wildcards are in play.

Each plot in the specified gallery includes *preview capsules* in its top-left corner. The capsules display the visualized data fields and transformation functions, which are indicated using bold capitalized text (*e.g.*, **MEAN**). Hovering over a capsule triggers an *encoding preview*, in which the shelves transiently update to show the specification of the selected plot and highlight corresponding fields. Encoding previews can help analysts understand the visual encodings used and help novice users learn how to construct particular types of charts.

Akin to a single focus view, the top-right corner for each plot in the specified gallery contains buttons for chart modification and bookmarking. In addition, the *specify* button (, intended to look like encoding shelves), allows users to assign a plot as the new focus view (C7). Like preview capsules, the specify button triggers an encoding preview upon mouse hover.

To aid comparison across charts (C4), Voyager 2 uses the global minimum and maximum values of a data field as its axis range by default even for aggregated plots, making axis

ranges for the same data field consistent across plots. Users can disable this behavior using the toolbar.

As view specification primarily involves selection of data fields, transformations and visual encodings, we provide the following kinds of wildcards to enable partial specification:

Wildcard fields let analysts construct views that treat data fields as free variables. Voyager 2 provides preset wildcard fields for *categorical*, *temporal*, and *quantitative fields* (Figure 7.1-D). These wildcards represent all fields of a particular data type. Analysts can simply drag-and-drop a wildcard field onto an encoding shelf to create multiple charts in parallel. For example, Figure 7.3 uses wildcard fields to produce a gallery of scatter plots involving all pairs of quantitative fields.

To create a wildcard for a specific set of fields, analysts can author a *custom wildcard field* by dragging desired fields to the wildcard list. Figure 7.1-E shows a custom wildcard field for *miles per gallon* and *weight in lbs*. To avoid incongruous views, Voyager 2 prevents analysts from creating custom wildcard fields containing multiple data types.

Wildcard functions allow simultaneous specification of data field transformations. Checking the wildcard checkbox in an encoding shelf's dropdown menu (Figure 7.5-B) enables a wildcard function. Once enabled, the radio buttons for each function become checkboxes, allowing analysts to select which functions to include (Figure 7.5-C). For example, selecting *none* and *mean* produces charts with raw data and mean summaries, as shown in Figure 7.5-D. The encoding shelf capsule (Figure 7.5-A) denotes the use of wildcard functions with "?".

Wildcard shelves enable ambiguous assignment of data fields to encoding channels. Analysts can drag a field or wildcard field onto a wildcard shelf (Figure 7.1-G). In Figure 7.6, an analyst uses a wildcard shelf to add the *origin* field to a scatterplot of *horsepower* and *miles per gallon*. To avoid redundant views (C8), Voyager 2 only produces distinctly different encodings. For

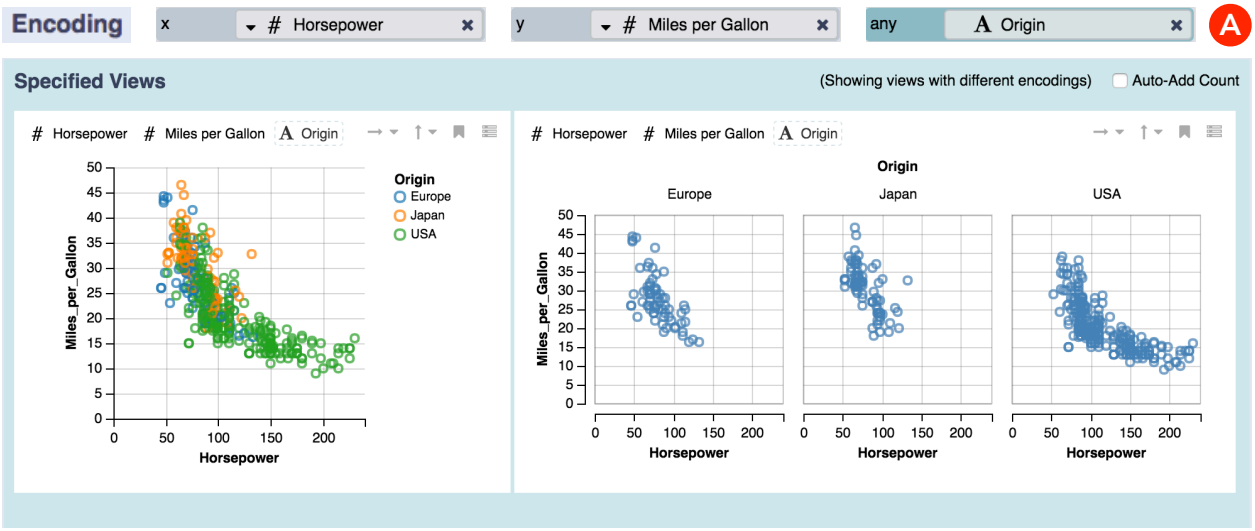


Figure 7.6: *Wildcard shelf* (A) allows users to consider alternative ways to encode *origin* (by using color or by faceting).

example, the output gallery only shows a column-based trellis plot and excludes a row-based trellis plot.

Applying multiple wildcards at the same time varies multiple parameters. For example, Figure 7.5 applies both a wildcard field and a wildcard function to see both distributions and mean values of each *quantitative field* conditioned on *origin* values. However, applying too many wildcards may produce a large number of views. Similar to Voyager, Voyager 2 prioritizes showing data variation over design variation (C1). When analysts use wildcard shelves together with wildcard fields or functions, Voyager 2 selects only the top-ranked encoding channel for the fields on the wildcard shelves.

7.3.3 Related Views to Promote Data Coverage

To promote data coverage and surface alternative ways to summarize and encode the data (C7), Voyager 2 presents multiple groups of *related views* based on the current focus view. Each group header (Figure 7.1-K, 7.1-L) denotes the type of suggestion. To avoid overwhelming

analysts with only one kind of suggestion (C8), Voyager 2 displays a limited number of views in each section and provides a "show more" button (Figure 7.4-A, bottom) for expanding the list. Users can also focus on a specific suggestion type via tab navigation. Each related view has interfaces similar to a view in the specified gallery. Voyager 2 supports the following types of related views:

Univariate summaries (Figure 7.2) show distributions for all data fields when no focus view is specified. Rather than starting with a blank screen, univariate summaries help users familiarize themselves with the different data fields (C1), following suggested practices for exploratory analysis [96]. To maintain consistency with the *data* panel (C4), these plots are ordered by data type and then by field name.

Summaries present aggregate plots to augment a focus view showing raw data. For example, Figure 7.1-K presents summary plots for the scatterplot between *horsepower* and *miles per gallon* in Figure 7.1-I. For quantitative fields, we calculate *mean* values or *bin* the fields to create histograms. To help users see distributions of discrete fields, we automatically add *count* to the summaries if the focus view has no quantitative fields. To determine top summary plots, Voyager 2 prefers views with fewer transformations to ease interpretation, and then sorts enumerated functions in the same order as the function drop-down dialog (Figure 7.5-B) to facilitate chart reading (C4).

Field suggestions show plots that contain one additional field. The goal is to help analysts consider other relationships that they might otherwise overlook (C1). To produce these views, Voyager 2 adds the suggested fields to the most effective channel still available (C3). To facilitate interpretation (C4), Voyager 2 keeps existing encodings of the focus view constant and groups views that add fields of the same data type together. For instance, Figure 7.1-L shows a group of recommended views that use the *color* channel to encode suggested categorical fields in the scatterplot between *horsepower* and *miles per gallon* (Figure 7.1-I). For consistency, the system orders the views by field name (C4). To avoid overplotting (C3),

Voyager 2 only provides field suggestions if there is an empty position shelf (*x* or *y*) available or if none of the non-positional channels (*color*, *size*, or *shape*) have been assigned.

Alternative encodings display other options for visualizing the same data (data fields and transformations identical to the focus view) when there are multiple effective encodings. For example, Figure 7.4 shows that analysts can encode *origin* using a partitioned trellis plot in addition to using a *color* encoding.

7.3.4 Refining Specifications of Related Views with Wildcards

Any section presented under *related views* can also be expressed using a *wildcard* encoding. To support refinement of suggestions (C7), each related view section header contains a specify button (). The button generates an *encoding preview* upon hover. Upon clicking the button, the encoding shelves are assigned a wildcard-based specification that generates the selected subset of related views. Thus, the specify button provides a mechanism for conducting focused analysis with a *collection* of recommended views, and may also help analysts learn how to use and interpret wildcards.

7.4 Enabling Partial Specification in Voyager 2 with CompassQL Queries

Both recommendations and specifications in Voyager 2 use the CompassQL visualization query language (Chapter 6) as an underlying representation. In this section, we will illustrate how interactions in Voyager 2 produce CompassQL queries, shown using the CompassQL JSON format.

A Focus View

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "field": "Miles_per_Gallon",
      "type": "quantitative"
    }]
  },
  "groupBy": "similarEncodings",
  "chooseBy": "effectiveness",
  "orderBy": "effectiveness"
}
```

B Wildcard Specification

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "?",
      "field": "?",
      "type": "quantitative"
    },{
      "channel": "?",
      "field": "?",
      "type": "quantitative"
    }]
  },
  "groupBy": “transformedFields”,
  "chooseBy": "effectiveness",
  "orderBy": "fieldOrder"
}
```

Figure 7.7: CompassQL queries for (A) the focus view in Figure 7.1 and (B) the wildcard specification in Figure 7.3.

7.4.1 Generating Queries for View Specifications

Similar to PoleStar (Section 5.6) and Tableau [131], Voyager 2 maintains a one-to-one mapping between the interface and the underlying specification. For example, Figure 7.7-A shows a query of the focus view in Figure 7.1-I. Wildcards in Voyager 2 also map directly to wildcards in CompassQL queries. For instance, using wildcard shelves and fields in Figure 7.3 produces a CompassQL query in which the corresponding encoding `channels` and `fields` are wildcards (Figure 7.7-B).

Voyager 2 prioritizes data variation (C1). If a specification has a wildcard field or function (Figure 7.7-B), Voyager 2 groups views with identical `transformedFields`. However, if there is no wildcard field or function, Voyager 2 shows design variations and groups views with `similarEncodings` (Figure 7.7-A).

For each group of similar views, Voyager 2 chooses an exemplar with the top perceptual `effectiveness` scores (C3). It then orders these exemplars to produce query results. To facilitate chart reading (C4), Voyager 2 first ranks them by `fieldOrder` if there is a wildcard field (Figure 7.7-B), and ranks them using the `functionOrder` scores if there is a wildcard function. It then ranks the exemplars based on their perceptual `effectiveness` (C3), as in Figure 7.7-A.

7.4.2 Generating Queries for Recommending Related Views

To provide related views, Voyager 2 uses the following methods to generate CompassQL queries from a focus chart.

To produce *univariate summaries* (Figure 7.2) when the focus view is empty, we use a fixed query template that includes a `count` field and a *wildcard field* with a *wildcard function* (Figure 7.8-A). To make each output view contain a group-by field for aggregating count, an expressiveness constraint implicitly limits the wildcard function to *bin* for a quantitative field, to time unit functions for a temporal field, or to *none* for a nominal field. Both fields are mapped to wildcard channels of a wildcard mark, letting the system pick appropriate encodings and orientation. Akin to Voyager, Voyager 2 prefers the y-axis for encoding nominal fields, resulting in horizontal labels that are easier to read. To avoid redundancy (C8), Voyager 2 clusters univariate summaries with identical `fields` and chooses an exemplar with the top `effectiveness` score for each field (C3). It then sorts the exemplars by `fieldOrder` (C4).

To generate *summaries* for an unaggregated focus view as in Figure 7.1-K, we add a wildcard function (`fn`), constrained to `mean` or `bin`, to each quantitative field (Figure 7.8-B). For each temporal field, we add a wildcard function with varying time units. The `mark` type, left as a wildcard, is automatically determined. The `autoAddCount` flag automatically aggregates the

A Univariate Summaries

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "?",
      "fn": "?",
      "field": "?",
      "type": "?"
    },
    {
      "channel": "?",
      "fn": "count",
      "field": "*",
      "type": "quantitative"
    }]
  },
  "groupBy": "fields",
  "chooseBy": "effectiveness",
  "orderBy": "fieldOrder"
}
```

B Related Summaries

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "fn": ["bin", "mean"],
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "fn": ["bin", "mean"],
      "field": "Miles_per_Gallon",
      "type": "quantitative"
    }]
  },
  "groupBy": "transformedFields",
  "chooseBy": "effectiveness",
  "orderBy": "functionOrder",
  "config": {"autoAddCount": true}
}
```

C Field Suggestions

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "x",
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "y",
      "field": "Miles_per_Gallon",
      "type": "quantitative"
    },{
      "channel": "?",
      "field": "?",
      "type": "nominal"
    }]
  },
  "groupBy": "fields",
  "chooseBy": "effectiveness",
  "orderBy": "fieldOrder"
}
```

D Alternative Encodings

```
{
  "spec": {
    "data": {"url": "cars.json"},
    "mark": "?",
    "encodings": [{
      "channel": "?",
      "field": "Horsepower",
      "type": "quantitative"
    },{
      "channel": "?",
      "field": "Miles_per_Gallon",
      "type": "quantitative"
    },{
      "channel": "?",
      "field": "Origin",
      "type": "nominal"
    }]
  },
  "groupBy": "similarEncodings",
  "chooseBy": "effectiveness",
  "orderBy": "effectiveness"
}
```

Figure 7.8: CompassQL queries for different kinds of related views: (A) *univariate summaries* in Figure 7.2, (B) *related summaries* in Figure 7.1-K, (C) *field suggestions* in Figure 7.1-L, and (D) *alternative encodings* in Figure 7.4-B.

count of records for plots with only discrete fields. Voyager 2 groups summary views with identical sets of `transformedFields` (C1), chooses exemplars based on `effectiveness` (C3), and sorts views with enumerated functions using the `functionOrder` scores (C4).

To provide *field suggestions* (Figure 7.1-L), we augment the focus view's specification with a wildcard field constrained to a fixed data type (Figure 7.8-C). To provide an appropriate level of detail for the added field, we apply a wildcard to constrain `fn` to *none*, *bin* or *mean* for quantitative fields, and to varying time units for temporal fields. Voyager 2 groups views with identical `fields` (C8), chooses exemplars with the highest `effectiveness` (C3), and sorts them by `fieldOrder` (C4).

To suggest *alternative encodings* (Figure 7.4-B), we replace the encoding *channels* and *mark* type with wildcards (Figure 7.8-D). To show distinctly different designs (C8), Voyager 2 clusters `similarEncodings` and excludes charts similar to the focus view from the query results. Per-cluster exemplars are chosen and sorted based on their perceptual `effectiveness` (C3).

7.5 Evaluation: Voyager 2 versus PoleStar

We conducted a user study to assess Voyager 2's ability to support both breadth- and depth-oriented analysis. We compared Voyager 2 with *PoleStar* (Section 5.6.1), a specification interface modeled on Tableau [131], a state-of-the-art tool for visual analysis. Our study design isolates *wildcards* and *related views* as the only difference between conditions. To this aim, we extended PoleStar to support filtering as well as *automatic mark* and *add to shelf* features [91]. To facilitate cross-study comparison between Voyager and Voyager 2, our design mirrors the previous study in Chapter 5, which compares Voyager and PoleStar. We hypothesized that Voyager 2 would lead to higher data field coverage than PoleStar, and that, unlike Voyager, Voyager 2 would enable analysts to flexibly drill-down and answer specific questions.

7.5.1 Study Design

Our study employed a 2 (interface) $\times$ 2 (dataset) mixed design. Each subject conducted two exploratory analysis sessions, each with a different, counterbalanced tool and dataset.

Datasets. To facilitate cross-study comparison, we reused the datasets from a prior study (Section 5.6): statistics about motion pictures ("movies") and a redacted version of FAA wildlife airplane strike records ("birdstrikes"). The movies dataset contains 3,201 records and 15 fields (7 nominal, 1 temporal, 8 quantitative). The birdstrikes dataset has 10,000 records and 14 fields (8 nominal, 1 geographic, 1 temporal, 4 quantitative).

Participants. We recruited 16 participants (11 female, 5 male), including 11 graduate students, 2 researchers, and 3 software engineers. All subjects had prior data analysis experience: all had used Excel, 9 had used Tableau or PowerBI, 13 had used Python/matplotlib, and 11 had used R/ggplot. All subjects had neither analyzed the study datasets before, nor had they used Voyager 2 or PoleStar. However, some subjects found basic interactions in both tools familiar due to their experience with Tableau. Each subject spent approximately 2 hours in our study. They received a $15 gift certificate as compensation.

Study Protocol. Before each session, we provided a 15-minute tutorial of the tool using the automobile dataset [110]. We then briefly introduced subjects to the experimental dataset. We asked participants to "comprehensively explore the data." We also asked them to bookmark insightful views and add text notes describing their rationale. To encourage accountability, subjects were told they would need to verbally summarize their insights at the end of each session, using their bookmarked views. We used a think-aloud protocol, asking participants to verbalize their thought process during the session. We did not ask participants to formulate specific questions before the session, as we did not want to bias them toward premature fixation on a specific set of questions. Subjects had 30 minutes to explore the dataset in each session, but could choose to end the session early once satisfied with their exploration.

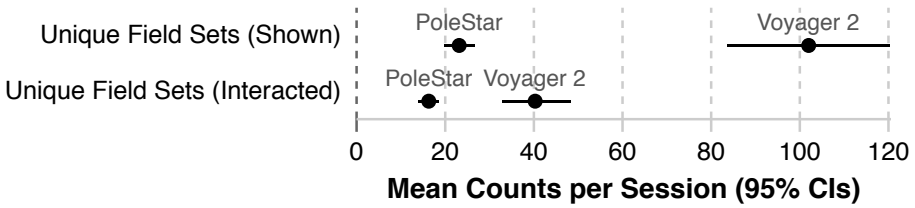


Figure 7.9: Mean counts and 95% CIs of unique field sets shown and interacted with. Users view and interact with more fields using Voyager 2.

We held all sessions in a laboratory setting. Participants ran both tools in Google Chrome on a Macbook Pro with a 15-inch retina display (2,880 $\times$ 1,980 pixels). After completing the analysis sessions, participants completed an exit survey.

Collected Data. An experimenter observed each session and took notes. We recorded audio and the screen for later review. Both visualization tools recorded interaction logs, including all input and application events. Finally, the exit survey included Likert scale ratings and subjects' rationales for their ratings.

7.5.2 Analysis of Usage Logs

We use linear mixed-effects models [26] to analyze usage log data. We include visualization tool and session order as fixed effects, and include intercept terms for dataset and subject as random effects (representing per-dataset and per-subject bias). Following common practice, to assess significance we use likelihood-ratio tests that compare a full model to a reduced model in which the fixed effect in question has been removed.

Voyager 2 promotes increased data field coverage. As in the previous Voyager study (Section 5.6), we want to assess the breadth of exploration. We consider the number and percentage of unique field sets both shown and interacted with. Following prior work [59, 72], we use mouse hover and interaction with a chart's toolbar as proxies to assess if a user has examined

a chart. This approximation provides a conversative estimate, as users may scrutinize charts without direct interaction.

In Figure 7.9, the visualization tool used significantly affects the number of unique field sets shown ($\chi^2(1) = 43.380$, $p < 0.001$) and interacted with ($\chi^2(1) = 26.999$, $p < 0.001$). On average, subjects were exposed to 102 unique field sets in Voyager 2 (over a baseline of 23 for PoleStar) and interacted with 41 unique field sets (over a baseline of 17). Comparing across studies, in prior work Voyager led to 3.2x more field sets seen and 1.5x more field sets interacted with on average. Here, the numbers for Voyager 2 are 4.4x and 2.4x: evidence that Voyager 2 similarly supports breadth-oriented analysis. While the prior study found an effect due to tool presentation order, we do not find such an effect in this study.

For the percentage of unique fields, Voyager 2 users were on average exposed to 98% of all fields in a dataset, over a baseline of 80% for PoleStar ($\chi^2(1) = 17.476$, $p < 0.001$), and interacted with 93% of fields versus a baseline of 79% ($\chi^2(1) = 10.644$, $p < 0.001$). In other words, subjects overlooked 20% of the fields in a dataset on average when using PoleStar, while only 7% were overlooked using Voyager 2. Moreover, 12/16 (75%) subjects interacted with at least 90% of all fields in Voyager 2, while only 5/16 (31%) did in PoleStar.

Bookmark rate is unaffected by visualization tool. Analyzing the count of bookmarked views, we find no effect due to tool ($\chi^2(1) = 0.381$, $p = 0.537$). Echoing the prior study in Section 5.6, we find that users bookmark views at similar rates, despite increased exposure from recommendations. We do not find an effect due to presentation order ($\chi^2(1) = 1.675$, $p = 0.196$).

Plots in related views are the most interacted with and bookmarked. From the total of 233 minutes that subjects interacting with charts in Voyager 2, they spent 109 minutes (46.8%) with related views, 41 minutes (17.7%) with views created with wildcards, and 83 minutes (35.5%) with views created without wildcards. Drilling down, 8/16 (50%) subjects spent the majority

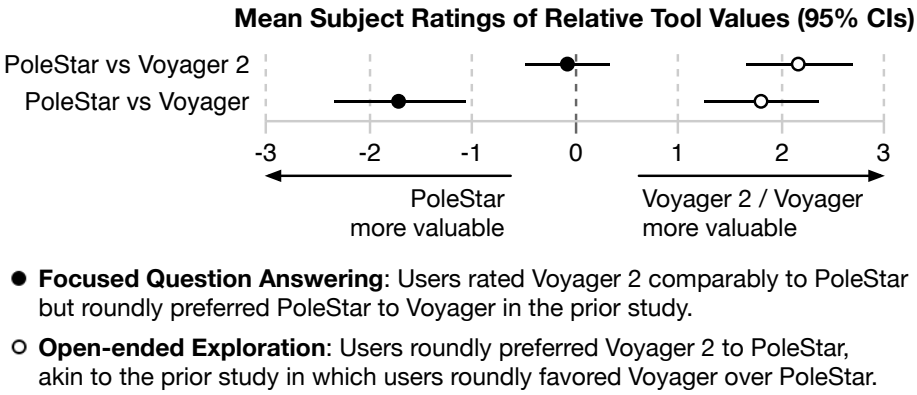


Figure 7.10: Mean subject ratings and 95% CIs of relative tool value for question answering and open-ended exploration (symmetric 7-point scale) from the Voyager 2 study and the prior Voyager study in Section 5.6. Voyager 2 has higher overall ratings than Voyager and PoleStar in terms of supporting *both* analysis phases.

of their time interacting with related views. Time spent interacting with either related views or wildcard views accounts for the majority of time for 14/16 (87.5%) subjects.

To assess if these interactions led to notable discoveries, we analyzed the source of bookmarked views. For 144 total charts bookmarked in Voyager 2, 62 (43.1%) are from related views, 32 (22.2%) are from views created with wildcards, and 50 (34.7%) are from views created without wildcards. Bookmarks from related views or from views created using wildcards account for the majority of bookmarks for 11/16 (68.8%) subjects. This result suggests that related views and wildcards contributed to finding interesting views.

7.5.3 Analysis of User Ratings

In the exit survey we asked subjects to reflect on their experiences and to provide ratings for both tools and features.

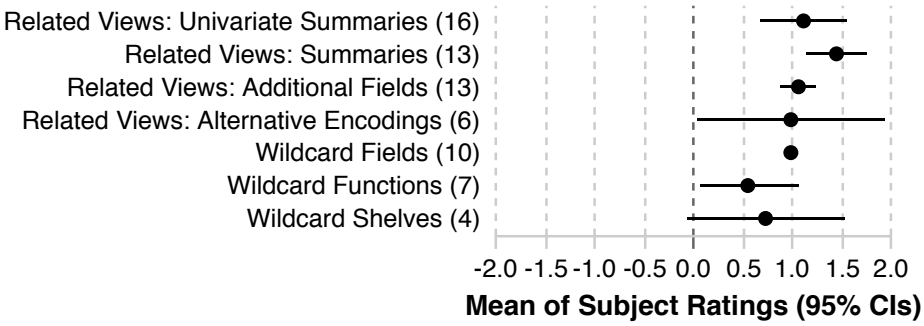


Figure 7.11: Mean usefulness ratings and 95% CIs for Voyager 2 features on a symmetric 5-point scale (-2 not useful, +2 very useful) show that subjects find both *related views* and *wildcards* features useful. Labels include the number of subjects who used and rated each feature.

Voyager 2 excels at exploration and supports focused analysis. Subjects rated the tools they found more valuable for exploration and question answering, using a symmetric 7-point scale (Figure 7.10). For exploration, 15 subjects find Voyager 2 "more" or "much more valuable", while only one finds PoleStar "somewhat more valuable" (μ = 2.19, σ = 0.981, t(15) = 8.919, p < 0.001). For question answering, no participant has a strong preference: 7 subjects are neutral while 5 and 4, respectively, find PoleStar or Voyager 2 "somewhat more valuable" (-1, +1) (μ = −0.063, σ = 0.772, t(15) = −0.324, p = 0.751). In the prior study (Section 5.6), subjects preferred Voyager for exploration, but strongly favored PoleStar for question answering. Compared to Voyager and PoleStar, Voyager 2 has higher overall ratings for supporting both analysis phases.

When asked how comprehensive they believed their analysis to be, subjects reported similar confidence levels for both tools (Voyager 2: μ = 0.375, σ = 1.258; PoleStar: μ = 0.375, σ = 1.628; W = 135.5, p = 0.786). Although Voyager 2 has more features, user ratings indicate comparable ease of use (Voyager 2: μ = 1.313, σ = 1.352; PoleStar: μ = 1.188, σ = 1.223; W = 117, p = 0.680).

Participants use wildcards less, but find both related views and wildcards helpful when used. Participants also rated the usefulness of various Voyager 2 features using a symmetric 5-point scale. Participants could give negative ratings if they found a feature "distracting", or indicate "did not use" if they did not use a certain feature in the session. As shown in Figure 7.11, subjects found the Voyager 2 features useful. Though used less frequently, wildcards were well-received. Subjects with more analysis expertise particularly appreciated the ability to explore multiple fields in a controlled fashion.

7.5.4 Participant Feedback: Balancing Automation & Control

In their free text comments, subjects described how Voyager 2 (especially *related views*) aided exploration and learning:

> *"I feel more confident using Voyager [2]. It helped me to learn. PoleStar feels scarier and like using the older SPSS tools. Voyager [2] seems more learner friendly."*

> *"Voyager 2 does all the summary charts for you, whereas I spent most of my time building similar summary charts in PoleStar."*

> *"The related view suggestion function in Voyager [2] accelerates exploration a lot."*

> *"I liked that Voyager [2] showed me what fields to include in order to see a specific graph. With PoleStar, I had to do a lot of trial and error and couldn't express what I wanted to see."*

Some participants also raised concerns for how recommendations might shape an analyst's thought process:

> *"These related views are so good but it's also spoiling that I start thinking less. I'm not sure if that's really a good thing."*

Subjects also commented on the use of *wildcards*:

> *"I found wildcards useful when I wanted a quick view to compare all the categorical or quantitative variables."*
>
> *"Creating my own wildcard field made it easier for me to compare the output graphs side by side."*
>
> *"I wasn't sure in advance sometimes if the graph I wanted to see should use Bin or Median/Mean, so I used the Wildcard Function to compare, pick, and learn for future use."*
>
> *"The wildcard seems to offer the same functionality as some of the related views but would give me more control over what I wanted to see. I think this would be really useful if I knew the tool well and had been using it for several weeks."*

One participant with a strong statistics background also noted that, though she made heavy use of related views, with more experience she would prefer the control afforded by wildcards.

7.5.5 Evaluation Summary

Together, our results indicate that Voyager 2's partial specification interfaces facilitate broader exploration, tool learning, and enable both serendipitous and controlled discovery. They also suggest the need for more study of these interfaces. Wildcards were viewed as powerful aids for analysis that might become increasingly useful with additional training. Related views were praised for accelerating analysis and suggesting otherwise overlooked directions, but also raise questions about the degree to which analysts might rely on them. Does the ready presence of relevant suggestions erode an analyst's independent thought process, and if so, to what ends?

7.6 Conclusion

We contribute Voyager 2, a visual analysis tool that combines manual and automatic chart specification in a single unified system. We introduce two partial specification interfaces: *wildcards* let users precisely vary the properties of a specification to generate multiple charts in parallel, while *related views* recommends visualizations relevant to the user's current focus. Both specifications and recommendations in Voyager 2 are represented using CompassQL [149], a visualization query language based on Vega-Lite [118]. Our controlled study evaluates this unified system approach. Comparing our results to a prior study [150], we find that Voyager 2 improves upon both a prior traditional specification interface (PoleStar) and a chart recommendation browser (Voyager) in terms of supporting *both* open-ended exploration and focused question answering.

To enable integration with existing data science ecosystems, we have also published Voyager 2 as an embeddable, open-source module. Based on this module, our colleagues Ji, Zhang, Brian Granger, and Saul Shanabrook has also built a JupyterLab extension [19] that allows users to load data from a Jupyter notebook to explore in Voyager 2 and export charts for sharing in the form of Vega-Lite specifications.

Voyager 2 is available as open-source software at `https://github.com/vega/voyager`.

8 Conclusion

Exploratory data analysis is a key activity for analysts to understand and extract value from their data. As exploratory analysis involves both open-ended exploration and focused question answering, an ideal analysis should cover both exploration breadth and analysis depth. However, existing exploratory analysis tools typically require a tedious chart specification process, preventing analysts from systematically exploring different aspects in the data. In addition, analysts may be blindsided by their own cognitive biases and prematurely fixate on specific questions or hypotheses. As a result, analysts without discipline and time may fail to achieve systematic breadth in their exploration and overlook important insights such as potentially confounding factors and data quality issues.

To address this challenge, this thesis contributes the design of exploratory analysis tools that complement manual specification with chart recommendation.

8.1 Review of Contributions

In Chapter 2, we conducted an interview study with 18 data analysts to better understand the current practice of exploratory data analysis. We characterize common exploration goals: *profiling* (assessing data quality) and *discovery* (gaining new insights). Though the EDA literature primarily emphasizes discovery, we observe that discovery only reliably occurs in

the context of open-ended analyses, whereas all participants engage in profiling across all of their analyses. We also describe the process and challenges of EDA highlighted by our interviews. We find that analysts must perform repetitive tasks (*e.g.,* examine numerous variables), yet they may have limited time or lack domain knowledge to explore data. Analysts also often have to consult other stakeholders and oscillate between exploration and other tasks, such as acquiring and wrangling additional data.

Based on the interviews, one major opportunity to mitigate observed challenges is facilitating rapid and systematic exploration with automation and guidance. The rest of this thesis addresses this opportunity by introducing a stack of systems for augmenting exploratory data analysis tools with chart recommendation. Specifically, this thesis presents new formal languages for chart specification and recommendation, and use them as foundations to build and study graphical interfaces that enable new forms of data exploration powered by chart recommendation.

The *Vega-Lite* visualization grammar (Chapter 4) serves as a formal model for specifying and reasoning about charts in our visualization recommender systems and user interfaces. Vega-Lite offers primitive building blocks for composing a broad range of visualizations in a concise JSON syntax. Vega-Lite provides concision by allowing specifications to omit low-level details and automatically inferring default values for these omitted details. With a concise JSON syntax, Vega-Lite enables both manual chart specification and programmatic generation in this thesis, and has served as a platform for developing other applications and research projects beyond this thesis.

With Vega-Lite as a formal model for chart specification, the *Voyager* visualization browser (Chapter 5) facilitates breadth-oriented data exploration with interactive navigation of recommended charts. In a user study comparing Voyager with a standard chart authoring tool, Voyager leads to increase data coverage and is preferred for open exploration tasks as it is less tedious and facilitates serendipitous discovery. However, participants still prefer a standard

chart authoring tool for question answering as they have the flexibility to build arbitrary plots to answer their questions. This result indicated the value of chart recommendation for open exploration, but also called for a unified tool that supports both manual chart specification and recommendation browsing.

The *CompassQL* query language (Chapter 6) provides a general-purpose framework for chart recommendation via queries over the space of visualizations. By combining a partial Vega-Lite specification with definitions of recommendation methods, a CompassQL query can describe a variety of existing chart recommendation approaches and serve as a unified representation for both chart specification and recommendation.

Building on CompassQL, *Voyager 2* (Chapter 7) blends manual and automated chart authoring to facilitate smooth gradations between open exploration and question answering in a single tool. Voyager 2 augments a standard chart authoring tool with two new partial specification interfaces: *wildcards* let users specify multiple charts in parallel, while *related views* suggest visualizations relevant to the currently specified chart. We find that Voyager 2 leads to increased data coverage compared to a traditional specification tool, while still allowing analysts to flexibly drill-down and answer specific questions.

8.2 Discussion and Future Directions

Taken together, this thesis contributes the design of systems that integrate chart recommendation into user-driven data exploration tools. As the key of exploratory data analysis is the analyst's flexibility to make analytical decisions based on what they discover in the data, our systems attempt to mitigate issues in exploratory data analysis by augmenting tools with recommendation while preserving user control to drive the analysis. Not only do the recommendations provided by these systems help reduce tedium and repetitive tasks in

exploration, but they also guide analysts with visualization and analysis practices as well as promote discovery of potentially relevant data.

Beyond supporting exploratory data analysis in this thesis, these systems have been released as open-source projects and widely adopted by both research and professional data science communities. In research, Vega-Lite has enabled a number of new research projects including an automatic model to reason about visualization similarity and sequencing [85] and natural language interfaces [5]. CompassQL was also used to generate training data for an analysis pipeline that reverse-engineers visual encodings from bitmap chart images [106]. A recent project called Draco [98] also builds on Vega-Lite and CompassQL to develop a chart recommender engine based on answer-set programming, making it easier to extend the constraints and learn ranking preferences. For professional data analysts, they can author charts with Vega-Lite via a number of wrappers such as Altair [11] and use Vega-Lite as a file format for saving and sharing charts. Via the Voyager 2 JupyterLab plugin [19], they can also easily explore data in Voyager 2, and transition between exploration and other analysis tasks such as data wrangling, within the Jupyter platform.

Building on the work presented in this thesis and its limitations, we see a number of future directions for exploratory data analysis tools and visualization recommendation.

8.2.1 More Sophisticated and Scalable Recommendation

This thesis presents CompassQL as a novel chart recommendation framework via queries over the space of visualizations. Based on this framework, we focus on designing user interfaces that augment exploration with recommendation, while preserving user control to steer the recommendation. However, as we focus on user interaction, one important area of future work is to design and evaluate more sophisticated chart recommendation.

For visual encodings, our chart recommenders only suggest simple Cartesian plots with primitive mark types (*e.g.,* bar, point, line, area). A direct extension would be to recommend more advanced statistical graphics. For example, box plots and layered charts can provide more statistical information, including variability and trend lines, in addition to the central tendency of a distribution.

From the Voyager 2 study (Chapter 7), we also learn that chart recommendation can help users learn as one participant said *"I feel more confident using Voyager 2. It helped me to learn"*. Extending our recommender systems to provide explanations for why certain designs are better can educate novices and improve visualization literacy, as well as make the recommendation more transparent to users.

For data field suggestion, the Voyager systems take a conservative approach by enumerating all fields and order them by their data types and names. This approach provides a predictable and consistent ordering. In addition, as the interview study in Chapter 2 suggests that analysts typically reduce number of fields to 10-20 fields before exploring, this approach could work well for exploring a sufficiently bounded number of fields. However, there remains room for improving recommendations. For specific exploration goals such as profiling, Voyager 2 may be extended to automatically detect and recommend variables with potential issues such as missing values or outliers [81].

Moreover, the interview study in Chapter 2 suggests that analysts primary choose data fields in their exploration primarily based on operational and domain knowledge (*e.g.,* whether a field is potentially important for their analysis or whether a field is redundant with other fields in the dataset). However, analysts often lack some knowledge and have to rely on other stakeholders to help them prioritize data fields in their exploration. To reduce this reliance, future chart recommendation may leverage some metadata about data importance to guide analysts with suggestions. How might the system collect this kind of information remains an

open problem. A few potential solutions include learning from user interactions and letting users directly annotate the datasets.

In terms of recommendation types, Voyager 2 currently presents four types of related view recommendation (univariate summaries, related summaries, alternative encodings, and field suggestions). While we find that these recommendations are useful for the user studies, it remains an open question how to design and evaluate recommendation types for data exploration. For example, while Voyager 2 only recommends univariate summaries in the initial state where the focus view is empty, future work may consider recommending univariate summaries of all variables involved in the focus view.

Another important issue for the Voyager systems is scalability. As the Voyager systems present multiple charts at the same time, they demand more computational power (for both transforming data and rendering charts) than a traditional authoring tool, which displays one chart at a time. One potential performance improvement is to detect common data transformation such as aggregation and binning among different charts and avoid redundant computation. In addition, future work may recommend more scalable visual encodings while working with large data. For example, the system may suggest density based plots such as histograms and binned scatterplots [37] instead of plotting all individual points.

8.2.2 Supporting Workflow Beyond Exploration Tasks

The Voyager 2 system supports smooth gradations between open-ended exploration and question answering in a single tool. However, our interview study in Chapter 2 also suggests that analysts often couple exploration with other analysis tasks such as data acquisition and wrangling, as well as consulting with and reporting to involved stakeholders.

One avenue to support other tasks such as data wrangling is to integrate exploration tool with existing analysis ecosystems. For Voyager 2, we have taken an initial step to support this

goal by releasing Voyager 2 as an embeddable module, allowing our colleagues to build the Voyager 2 for JupyterLab. Thus, analysts can wrangle data in JupyterLab (*e.g.*, with Python and the Pandas data frame [95]) before exploring data with Voyager 2.

Nevertheless, the interview study also notes the challenge for the navigation and search of analysis history. This challenge is even more critical in the context of the Voyager systems since users can explore more plots, and thus may have more need to revisit their exploration history. Future work may investigate how to visualize branches of exploration history to help analysts see an overview of what they have explored and revisit unexplored directions.

8.2.3 Better Understand the Effects of Automation in Data Exploration

The Voyager 2 study in Chapter 7 shows that Voyager 2's unified system that blends manual and automated chart specification can facilitate more rapid and systematic exploratory analysis. However, the benefits of automation may not come without risk as one study participant noted:

> *"These related views are so good but it's also spoiling that I start thinking less. I'm not sure if that's really a good thing."*

Thus, another important research is to further study the effects of automation in data analysis. On one hand, automation can help analysts focus on interpreting the data rather than executing routine tasks. On the other hand, the key of exploratory analysis is the analysts' flexibility to make analytical decisions based on what they discover in the data. Excessive automation and suggestion may interrupt the analysts' exploration flow [49], or lead them to accept recommendations and short-circuit their own critical thinking.

For Voyager 2, an important question is whether features with varying degree of automation like wildcards and related views would lead analysts to be complacent [101] and think less, or

bias them toward spurious discovery [154]. The study in Chapter 7 focuses on Voyager 2's unified system and thus only evaluates the system as a whole. However, detailed studies of isolated aspects of the system remain an important future work. Longitudinal studies might help us better understand how analysts learn and apply partial specification interfaces in their own work. By recording and modeling longer-term analysis activities, we might characterize analytic strategies and gain more insight into the benefits and potential drawbacks of introducing increased automation into the data exploration process. Insights from such studies may help us ensure that future data analysis tools will have the right balance between automation and user control.

8.2.4 Visualization Recommendation for Other Interfaces

This thesis introduces new formal languages as foundations for chart specification and recommendation, and use them to develop graphical user interfaces for exploratory data analysis. However, the foundations and techniques for chart recommendation presented in this thesis can also be applied to other kinds of interfaces.

As we found from the interview study in Chapter 2, plotting APIs such as ggplot2 [144] and Altair [11] remain one of the primary tools for chart authoring in data exploration and analysis. Using our chart recommendation infrastructure, we can augment these APIs with design "linting" and suggest better design alternatives. For example, a plotting library can throw a warning when a user encodes data on a log scale with bar marks and suggest the user to consider using a point mark instead.

Beyond desktop-based graphical interfaces, natural language interfaces have received growing interests in the recent years, both in research [55,128,129,132] and in industry [17,70,122]. Some ongoing work on natural language interfaces have already used Vega-Lite as the underlying chart representation. One promising approach to develop a natural language

interface for chart authoring is mapping a natural language command to a visualization query language such as CompassQL. The underlying recommender engine can then produce results that satisfy the given query. As natural language commands will often include information about the users' intended tasks, one important research is incorporating task models (*e.g.*, [22]) into the query language and recommender engine [38,57,155]. For example, ranking metrics should take task-based effectiveness of visual encodings [115] into their consideration.

8.2.5 Automated Design of Interactive Dashboard

Beyond authoring and exploring individual charts, dashboard creation is another common use case of data visualization tools [116]. A promising direction is to automate the design of interactive multi-view displays based on a user-provided intent. While Vega-Lite's grammar for view composition and interaction (Section 4.2) is already useful as a basis for this research, there remain a number of open challenges to enable this application.

First, as our visualization query language and interfaces are currently limited to single static displays, future work needs to extend the query language and interface with mechanisms for expressing queries of interactive, multi-view displays.

Moreover, while we have taken initial steps to support reading of multiple charts in context in Voyager and Voyager 2, most of existing visualization design criteria mainly focus on the design of a single chart. To evaluate dashboard designs, future work must study new guidelines for interactive, multi-view displays. For example, effectiveness metrics should consider consistency between visual encodings of subplots in a multi-view display in addition to the effectiveness of each individual subplot [107].

Finally, as we go from single static charts to interactive dashboards, the space of possible designs will grow exponentially. We need an efficient way to develop rankings that can scale

to cover this design space. The recent Draco system [98] applies answer-set programming and learning to rank approaches to develop an extensible recommendation model. However, an efficient strategy to collect training data for this large design space remains an open problem. Existing model training techniques such as active learning [123] and machine teaching [127] may provide some clues for this challenge.

8.3 Concluding Remarks

In this thesis, we present foundations for chart specification and recommendation, and apply these foundations to facilitate rapid and systematic exploratory data analysis by augmenting data exploration tools with chart recommendation. Beyond this thesis, these systems have enabled many new applications and research projects. Yet there remain many directions that these ideas and systems can be applied. We hope that this thesis will continue to influence the design of future systems for data visualization and analysis. All the systems presented in this thesis are available as open-source software at `https://vega.github.io/`.

Bibliography

[1] Matplotlib documentation. http://matplotlib.org/.

[2] Spotfire. http://spotfire.tibco.com/.

[3] Spotfire recommendations. http://spotfire.tibco.com/recommendations.

[4] Mckinsey quarterly, 2009.

[5] A conversational agent for data science., 2016. https://hackernoon.com/a-conversational-agent-for-data-science-4ae300cdc220.

[6] vegalite: R ggplot2 "bindings" for vega-lite., 2016. https://github.com/hrbrmstr/vegalite.

[7] Vegalite.jl: Julia bindings to vega-lite., 2016. https://github.com/fredo-dedup/VegaLite.jl.

[8] vegaliter: A vega-lite htmlwidget for r., 2016. https://github.com/timelyportfolio/vegaliteR.

[9] Vegas: The missing matplotlib for scala + spark., 2016. https://github.com/vegas-viz/Vegas.

[10] Vizard: Magic visualization., 2016. https://github.com/yieldbot/vizard.

[11] Altair: Declarative visualization in python., 2018. https://altair-viz.github.io.

[12] Apache arrow. https://arrow.apache.org/, 2018. Accessed on Aug 1, 2018.

[13] Exploring data with vega-lite, 2018. https://beta.observablehq.com/@mbostock/exploring-data-with-vega-lite.

[14] Google data studio. https://datastudio.google.com/u/0/, 2018. Accessed on Aug 1, 2018.

[15] Jupyterlab. https://github.com/jupyterlab/jupyterlab, 2018. Accessed on Aug 1, 2018.

[16] Microsoft powerbi. https://powerbi.microsoft.com/, 2018. Accessed on Aug 1, 2018.

[17] Q&a in power bi service and power bi desktop., 2018. https://docs.microsoft.com/en-us/power-bi/power-bi-q-and-a.

[18] Tableau prep. https://www.tableau.com/products/prep, 2018. Accessed on Aug 1, 2018.

[19] Voyager extension for jupyterlab. https://github.com/altair-viz/jupyterlab_voyager, 2018. Accessed on Aug 1, 2018.

[20] A dramatic tour through python's data visualization landscape., October, 2016. https://dsaber.com/2016/10/02/a-dramatic-tour-through-pythons-data-visualization-landscape-including-ggplot-and-altair/.

[21] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *The VLDB Journal-The International Journal on Very Large Data Bases*, 24(4):557–581, 2015.

[22] Robert Amar, James Eagan, and John Stasko. Low-level components of analytic activity in information visualization. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 111–117. IEEE, 2005.

[23] Anushka Anand and Justin Talbot. Automatic selection of partitioning variables for small multiple displays. *Visualization and Computer Graphics, IEEE Transactions on*, 22(1):669–677, 2016.

[24] Natalia Andrienko and Gennady Andrienko. *Exploratory analysis of spatial and temporal data: a systematic approach*. Springer Science & Business Media, 2006.

[25] Zan Armstrong and Martin Wattenberg. Visualizing statistical mix effects and simpson's paradox. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis)*, 20(12):2132–2141, 2014.

[26] Dale J. Barr, Roger Levy, Christoph Scheepers, and Harry J. Tily. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language*, 68(3):255–278, 2013.

[27] Andrea Batch and Niklas Elmqvist. The interactive visualization gap in initial exploratory data analysis. *IEEE transactions on visualization and computer graphics*, 24(1):278–287, 2018.

[28] Richard A Becker, William S Cleveland, and Ming-Jen Shyu. The visual design and control of trellis display. *Journal of computational and Graphical Statistics*, 5(2):123–155, 1996.

[29] John T Behrens. Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131, 1997.

[30] John T Behrens and Chong-ho Yu. Exploratory data analysis. *Handbook of psychology*, 2:33–64, 2003.

[31] Jacques Bertin. *Semiology of graphics: diagrams, networks, maps*. University of Wisconsin press, 1983.

[32] Enrico Bertini, Andrada Tatu, and Daniel Keim. Quality metrics in high-dimensional data visualization: an overview and systematization. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis)*, 17(12):2203–2212, 2011.

[33] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for bisualization. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis)*, 15(6):1121–1128, 2009.

[34] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Comp. Graphics*, 17(12):2301–2309, 2011.

[35] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747. ACM, 2006.

[36] Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[37] Daniel B Carr, Richard J Littlefield, WL Nicholson, and JS Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[38] Stephen M. Casner. Task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics (TOG)*, 10(2):111–151, 1991.

[39] Chris Chatfield. The initial examination of data. *Journal of the Royal Statistical Society. Series A (General)*, pages 214–253, 1985.

[40] Chris Chatfield. Exploratory data analysis. *European journal of operational research*, 23(1):5–13, 1986.

[41] Chris Chatfield. *Problem solving: a statistician's guide*. Chapman and Hall/CRC, 1995.

[42] Ed Huai-hsin Chi and John T Riedl. An operator interaction framework for visualization systems. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 63–70. IEEE, 1998.

[43] George Chin Jr, Olga A Kuchar, and Katherine E Wolf. Exploring the analytical processes of intelligence analysts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 11–20. ACM, 2009.

[44] Bum chul Kwon, Brian Fisher, and Ji Soo Yi. Visual analytic roadblocks for novice investigators. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 3–11. IEEE, 2011.

[45] William S. Cleveland. *The elements of graphing data*, volume 2. Wadsworth Advanced Books and Software Monterey, CA, 1985.

[46] William S. Cleveland. *The Collected Works of John W. Tukey: Graphics 1965-1985*, volume 5. CRC Press, 1988.

[47] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[48] Dianne Cook, Deborah F Swayne, and Andreas Buja. *Interactive and dynamic graphics for data analysis: with R and GGobi*. Springer Science & Business Media, 2007.

[49] Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *Harper-Perennial, New York*, 39, 1997.

[50] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007.

[51] Stephen Few. *Now you see it: simple visualization techniques for quantitative analysis*. Analytics Press, 2009.

[52] James J Filliben and Alan Heckert. Exploratory data analysis. *Engineering Statistics Handbook, Internet, National Institute of Standards and Technology*, 2005.

[53] Danyel Fisher and Miriah Meyer. *Making Data Visual: A Practical Guide to Using Visualization for Insight*. O'Reilly, 2017.

[54] Danyel Fisher, Igor Popov, Steven Drucker, et al. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1673–1682. ACM, 2012.

[55] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 489–500. ACM, 2015.

[56] Andrew Gelman. Exploratory data analysis for complex models. *Journal of Computational and Graphical Statistics*, 13(4):755–779, 2004.

[57] David Gotz and Zhen Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 315–324, 2009.

[58] David Gotz and Michelle X Zhou. Characterizing users' visual analytic activity for insight provenance. *Information Visualization*, 8(1):42–55, 2009.

[59] Spence Green, Jeffrey Heer, and Christopher D. Manning. The efficacy of human post-editing for language translation. In *Proc. ACM Human Factors in Computing Systems (CHI)*, 2013.

[60] Philip Jia Guo. *Software tools to facilitate research programming*. PhD thesis, Stanford University Stanford, CA, 2012.

[61] Robert M Haralick and Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

[62] Marti Hearst. *Search user interfaces*. Cambridge University Press, 2009.

[63] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics*, 14(6), 2008.

[64] Jeffrey Heer and Ben Shneiderman. Interactive dynamics for visual analysis. *Communications of the ACM*, 55(4):45–54, April 2012. `https://idl.cs.washington.edu/papers/interactive-dynamics`.

[65] Jeffrey Heer, Frank Van Ham, Sheelagh Carpendale, Chris Weaver, and Petra Isenberg. Creation and collaboration: Engaging new audiences for information visualization. In *Information Visualization*, pages 92–133. Springer, 2008.

[66] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. Online aggregation. In *Acm Sigmod Record*, volume 26, pages 171–182. ACM, 1997.

[67] Harold V Henderson and Paul F Velleman. Building multiple regression models interactively. *Biometrics*, pages 391–411, 1981.

[68] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[69] David C Hoaglin, Frederick Mosteller, and John W Tukey. Understanding robust and exploratory data anlysis. *Wiley Series in Probability and Mathematical Statistics, New York: Wiley, 1983, edited by Hoaglin, David C.; Mosteller, Frederick; Tukey, John W.*, 1983.

[70] Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE transactions on visualization and computer graphics*, 24(1):309–318, 2018.

[71] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 159–166, 1999.

[72] Jeff Huang, Ryen White, and Georg Buscher. User see, user point: gaze and cursor alignment in web search. In *Proc. ACM Human Factors in Computing Systems (CHI)*, 2012.

[73] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 277–281. ACM, 2015.

[74] Petra Isenberg, Anthony Tang, and Sheelagh Carpendale. An exploratory study of visual information analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1217–1226. ACM, 2008.

[75] TJ Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, 2007.

[76] LV Jones. *The Collected Works of John W. Tukey: Philosophy and Principles of Data Analysis 1949-1964*, volume 3. CRC Press, 1986.

[77] LV Jones. *The Collected Works of John W. Tukey: Philosophy and Principles of Data Analysis 1965-1986*, volume 4. CRC Press, 1987.

[78] Daniel Kahneman and Shane Frederick. Representativeness revisited: Attribute substitution in intuitive judgment. *Heuristics and biases: The psychology of intuitive judgment*, 49:81, 2002.

[79] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

[80] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. In *IEEE Visual Analytics Science & Technology (VAST)*, 2012.

[81] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proc. Advanced Visual Interfaces (AVI)*, pages 547–554. ACM, 2012.

[82] Youn-ah Kang and John Stasko. Characterizing the intelligence analysis process: Informing visual analytics design through a longitudinal field study. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 21–30. IEEE, 2011.

[83] Daniel A Keim, Florian Mansmann, Jörn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 9–16. IEEE, 2006.

[84] Alison Kidd. The marks are on the knowledge worker. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 186–191. ACM, 1994.

[85] Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer. Graphscape: A model for automated reasoning about visualization similarity and sequencing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2628–2638. ACM, 2017.

[86] Heidi Lam. A framework of interaction costs in information visualization. *IEEE transactions on visualization and computer graphics*, 14(6), 2008.

[87] Gaea Leinhardt and Samuel Leinhardt. Chapter 3: Exploratory data analysis: New tools for the analysis of empirical data. *Review of research in education*, 8(1):85–157, 1980.

[88] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 2014.

[89] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[90] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[91] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis)*, 13(6):1137–1144, 2007.

[92] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

[93] Gary Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[94] Joe Marks, Brad Andalman, Paul A. Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400. ACM Press/Addison-Wesley Publishing Co., 1997.

[95] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011.

[96] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics.* WH Freeman/Times Books/Henry Holt & Co, 1989.

[97] Stephan Morgenthaler. Exploratory data analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):33–44, 2009.

[98] Dominik Moritz, Chenglong Wang, Greg Nelson, Halden Lin, Adam Smith, Bill Howe, and Jeff Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. In *IEEE Vis (Proc. InfoVis)*, 2018.

[99] Felix Naumann. Data profiling revisited. *ACM SIGMOD Record*, 42(4):40–49, 2014.

[100] Raymond S Nickerson. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175, 1998.

[101] Raja Parasuraman and Dietrich H Manzey. Complacency and bias in human use of automation: An attentional integration. *Human factors*, 52(3):381–410, 2010.

[102] Fernando Perez and Brian E Granger. Project jupyter: Computational narratives as the engine of collaborative data science. *Retrieved September*, 11:207, 2015.

[103] Jeffrey M Perkel. Data visualization tools drive interactivity and reproducibility in online publishing. *Nature*, 554(7690):133–134, 2018.

[104] Daniel B. Perry, Bill Howe, Alicia M.F. Key, and Cecilia Aragon. VizDeck: Streamlining exploratory visual analytics of scientific data. In *Proc. iSchool Conference*, 2013.

[105] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4. McLean, VA, USA, 2005.

[106] Jorge Poco and Jeffrey Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, volume 36, pages 353–363. Wiley Online Library, 2017.

[107] Zening Qu and Jessica Hullman. Keeping multiple views consistent: Constraints, validations, and exceptions in visualization authoring. *IEEE transactions on visualization and computer graphics*, 24(1):468–477, 2018.

[108] Eric D Ragan, Alex Endert, Jibonananda Sanyal, and Jian Chen. Characterizing provenance in visualization and data analysis: an organizational framework of provenance types and purposes. *IEEE transactions on visualization and computer graphics*, 22(1):31–40, 2016.

[109] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[110] Ernesto Ramos and David Donoho. Asa data exposition dataset, 1983. `http://statcomputing.org/dataexpo/1983.html`.

[111] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[112] Steven F. Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 112–117. ACM, 1994.

[113] Steven F Roth and Joe Mattis. Automating the presentation of information. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 90–97. IEEE, 1991.

[114] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 32. ACM, 2018.

[115] Bahador Saket, Alex Endert, and Cagatay Demiralp. Task-based effectiveness of basic visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[116] Alper Sarikaya, Michael Correll, Lyn Bartram, Melanie Tory, and Danyel Fisher. What do we talk about when we talk about dashboards? *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[117] Ali Sarvghad, Melanie Tory, and Narges Mahyar. Visualizing dimension coverage to support exploratory analysis. *IEEE transactions on visualization and computer graphics*, 23(1):21–30, 2017.

[118] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2017. https://idl.cs.washington.edu/papers/vega-lite.

[119] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 22(1):659–668, 2016.

[120] Howard J Seltman. Experimental design and analysis. *Online at:* http://www.stat.cmu.edu/hseltman/309/Book/Book.pdf, 2012.

[121] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization*, 4(2):96–113, 2005.

[122] Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 365–377. ACM, 2016.

[123] Burr Settles. Active learning literature survey. 2010. *Computer Sciences Technical Report*, 1648, 2014.

[124] Ben Shneiderman. Dynamic queries for visual information seeking. *Software, IEEE*, 11(6):70–77, 1994.

[125] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[126] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. 2017.

[127] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, et al. Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742*, 2017.

[128] Arjun Srinivasan and John Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *Proceedings of EuroVis*, volume 17, pages 55–59, 2017.

[129] Arjun Srinivasan and John Stasko. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE transactions on visualization and computer graphics*, 24(1):511–521, 2018.

[130] Stanley Smith Stevens. On the theory of scales of measurement, 1946.

[131] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[132] Yiwen Sun, Jason Leigh, Andrew E. Johnson, and Sangyoon Lee. *Articulate*: A semi-automated model for translating natural language queries into meaningful visualizations. In *Smart Graphics*, pages 184–195, 2010.

[133] Edward R Tufte. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.

[134] J. W. Tukey and P. A. Tukey. Computer graphics and explaoratory data analysis: An introduction. In *Proceedings of the Sixth Annual Conference and Exposition: Computer Graphics*, 1985.

[135] John W Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.

[136] John W Tukey. We need both exploratory and confirmatory. *The American Statistician*, 34(1):23–25, 1980.

[137] Stef van den Elzen and Jarke J van Wijk. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum*, 32(3pt2):191–200, 2013.

[138] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. SeeDB: Efficient data-driven visualization recommendations to support visual analytics. *VLDB 2015*, 8(13):2182–2193, 2015.

[139] Declarative visualization for Elm. `https://github.com/gicentre/elm-vega`, January 2018.

[140] Paul F Velleman and David C Hoaglin. *Applications, basics, and computing of exploratory data analysis*. Duxbury Press, 1981.

[141] Colin Ware. *Information visualization: perception for design*. Elsevier, 2012.

[142] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

[143] Ryen W. White and Resa A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.

[144] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2009.

[145] Hadley Wickham and Garrett Grolemund. *R for data science: import, tidy, transform, visualize, and model data*. " O'Reilly Media, Inc.", 2016.

[146] Leland Wilkinson. *The Grammar of Graphics*. Springer, 2005.

[147] Leland Wilkinson, Anushka Anand, and Robert L. Grossman. Graph-theoretic scagnostics. In *IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis)*, volume 5, page 21, 2005.

[148] Graham Wills and Leland Wilkinson. Autovis: automatic visualization. *Information Visualization*, 9(1):47–69, 2010.

[149] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Towards a general-purpose query language for visualization recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. ACM, 2016. https://idl.cs.washington.edu/papers/compassql.

[150] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2016. https://idl.cs.washington.edu/papers/voyager.

[151] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2648–2659. ACM, 2017. https://idl.cs.washington.edu/papers/voyager2.

[152] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 401–408, 2003.

[153] Ji Soo Yi, Youn ah Kang, John T Stasko, Julie A Jacko, et al. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization & Computer Graphics*, (6), 2007.

[154] Emanuel Zgraggen, Zheguang Zhao, Robert Zeleznik, and Tim Kraska. Investigating the effect of the multiple comparisons problem in visual analysis. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 479. ACM, 2018.

[155] Michelle X. Zhou and Min Chen. Automated generation of graphic sketches by example. In *IJCAI*, volume 3, pages 65–71, 2003.

A List of Built-in Constraints in CompassQL

In this appendix chapter, we provide a list of expressiveness constraints [90] in CompassQL (Chapter 6). These constraints are configurable; users can disable certain constraints.

A.1 Encoding constraints

CompassQL considers the following constraints for the relationships among the properties of an encoding channel.

- *The function should be supported by the data type.* For example, ordinal fields can support *median* but not *mean*; *bin*ning can only be applied to quantitative fields; *time unit* can only be applied to temporal fields.

- *The encoding channel should support the type of the field.* For example, *x*, *y*, *color* can support both partition fields (*e.g.,* nominal, ordinal, binned quantitative, discretized temporal) and continuous fields. Meanwhile, *row*, *column*, and *shape* can only serve as partition fields while *size* is inappropirate for nominal fields.

- *A particular time unit should be applied only if they produce unique values.* For example, if a temporal field only contains year information, it is not useful to enumerate different month, day, hours and minutes.

- *The scale type should support the specified scale properties.* For example, *pow* property is only applicable for log / power / square root scales. The include *zero* property is not applicable for log scale.

- *The field's primitive type should support the data type.* For example, a string field can be a nominal or ordinal field, but not quantitative. Only a date field can be a temporal field.

- *A field's cardinality should not exceed channel's limit.* For example, there are 6 unique shape palettes in Vega-Lite. Thus, the shape channel should not encode a field with higher cardinality.

- *Scale type should match data type.* For example, *log* and *power* scales are only applicable for quantitative fields.

A.2 Specification Constraints

CompassQL considers the following constraints for relationships between all properties of a specification.

- *No repeated encoding channel.* Each channel except *detail* in a Vega-Lite specification can be mapped to only one field.

- *All required channels for the specified mark should be specified.* For example, *line* and *area* marks require both *x* and *y* channel to be encoded. As we derive this constraint from the original Compass engine, Table 5.2 in Chapter 5 lists all supported channels for each mark.

- *Each encoding channel should be supported by the mark type.* For example, the *bar* mark does not support the *shape* channel. As we derive this constraint from the original Compass engine, Table 5.2 in Chapter 5 lists all supported channels for each mark.

- *Plot should have appropriate data types for mark.* For example, a *bar* is not appropriate for raw (unaggregated) plots with quantitative fields on both x- and y-axis.

- *Do not use bar, line, or area to visualize raw plots if they lead to occlusion.* Only use *bar*, *line*, or *area* for raw plots if there are only one continuous values in each partition. Otherwise, *point* and *tick* marks are more appropriate.

- *Do not recommend a bar mark if its length axis does not start at zero or is a log scale.* If not started at zero, the length of the bars might lead to a misleading ratio comparison.

- *Only apply auto-add-count if there is no continuous field.* In other words, only automatically add a *count* field if there are only discrete fields (nominal or ordinal fields, or discretized continuous fields) in the plot.

- *Do not map field to size channel with bar mark.* Stacked bar chart with varying width of bar is not a general practice, so we avoid it.

- *Avoid over-encoding.* Unless specified by the user, do not use multiple encoding channels to encode the same field.

- *Do not use non-positional channels unless all positional channels are used.* Positional channels (*x* and *y*) are the most effective channels. Therefore, 1D plots with non-positional channels such as colored dot plots are avoided.

- *Aggregate plot should not use raw continuous field as group by values.* To serve as a partition field, a continuous field should be discretized.

- *Do not use detail channel with raw plot.* Adding an additional level of detail to a raw (unaggregated) plot has no effect. Thus, it should not be a part of recommendation.
- *Stacked plot should use summative aggregation such as sum, count, or distinct.* Stacked plot that aggregates each part of the stacked bar with non-summative aggregation such as mean or median can be misleading.