INTERACTIVE SYSTEMS FOR DATA TRANSFORMATION AND ASSESSMENT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sean Kandel
June 2013

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Jeffrey Heer, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Patrick Hanrahan

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andreas Paepcke

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

In spite of advances in technologies for processing and visualizing data, analysts still spend an inordinate amount of time diagnosing data quality issues and manipulating data into a usable form. This process often constitutes the most tedious and time-consuming aspect of analysis. This dissertation contributes novel techniques for coupling automated routines with interactive interfaces to enable more rapid data transformation and quality assessment.

In this dissertation, we first present an interview study with enterprise data analysts. We characterize the process of industrial data analysis, document how organizational features of an enterprise impact analysis, describe recurring pain points, and discuss design implications for visual analysis tools.

Next we introduce Wrangler, an interactive system for creating data transformation scripts. Wrangler combines direct manipulation of visualized data with automatic inference of relevant transforms, enabling analysts to iteratively explore the space of applicable operations and preview their effects. We present user study results showing that Wrangler significantly reduces specification time and promotes the use of robust, auditable transforms instead of manual editing. Underlying the Wrangler interface is a declarative data transformation language that supports code-generation of executable code in a variety of runtime platforms.

For large data sets, an analyst can build and test a script on a sample of data before applying the script to the entire data set. Often times, errors or other anomalies will appear in the data set that did not appear in the sample. We introduce and evaluate two methods to aid more rapid debugging of large-scale transformation scripts. *Surprise-based anomaly detection* applies a model to classify output records as exceptions. *Rule-based transform disambiguation* generates example records to help analysts refine transformation scripts

before applying them.

After transforming a data set, an analyst often inspects the result for other data quality issues. We present Profiler, a visual analytic tool for assessing data quality issues. We present Profiler's architecture, including modular components for custom data types, anomaly detection routines and summary visualizations. The system contributes novel methods for integrated statistical and visual analysis, automatic view suggestion, and scalable visual summaries that support real-time interaction entirely in the browser with millions of data points.

Taken together, this dissertation contributes novel methods for integrating automated routines with interaction and visualization techniques to improve the efficiency and scale at which data analysts can work.

Acknowledgements

I was particularly fortunate to work with many amazing mentors during my time at Stanford. Jennifer Widom first sparked my interests in databases and research. Hector Garcia-Molina introduced me to the challenge of designing interfaces for data manipulation. Pat Hanrahan offered novel ways to approach my research and fascinating insights into the practice of visual analytics. Joe Hellerstein's work influenced much of my own and his ideas shaped much of my thinking. Andreas Paepcke was a sounding-board, friend, and creative inspiration throughout all of my projects.

I thank my advisor Jeffrey Heer. Through his example and guidance, I have grown as a writer, hacker, presenter and researcher. His contributions to my thinking and my career are immeasurable. Working with him has focused and motivated me for nearly four years; a feat once thought impossible.

I had the incredible opportunity to collaborate with members of the Vis Group and InfoLab. In particular, I thank Parag Agrawal, Anant Bhardwaj, Jason Chuang, Anish Das Sarma, Philip Guo, Paul Heymann, Robert Ikeda, Sanjay Kairam, Diana MacLean, Aditya Parameswaran, Justin Talbot, Petros Venetis, Steven Whang, and Andrea Zvinakis.

I am grateful to the friends who have tolerated me over the last few years. To the Kelman, Mahdavi, Martinez, and Seng families for their warmth and generosity. To Sarah and the inhabitants of 66 Mallorca for their support, poor judgment, and bad decisions. And to Lydia for her endless encouragement, enthusiasm, patience and wake up calls.

I thank my sister Jasmyn and brothers Tyler, Bryan, and Sandy. Their talent, creativity, vision, and curiosity inspire and motivate me.

Finally, I dedicate this thesis to my parents Stephen and Genevieve. I hope one day to give someone all that they have given to me. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The scale and availability of digital data is increasing exponentially [29]. This increase has been powered in part by continuing decreases in cost for data storage and computational processing. Taken together, these trends offer the promise of increased insights into many fields including finance, medicine, and our government.

However, the generation, collection and dissemination of data — including web, government, and sensor data — often outpaces our ability to analyze and make sense of it. In many cases, the bottleneck in performing analysis and extracting insights from data is the availability of skilled data analysts. As Hal Varian, Chief Economist at Google said in 2009,

> Now we really do have essentially free and ubiquitous data. So the complimentary scarce factor is the ability to understand that data and extract value from it. [1]

Moreover, the McKinsey Global Institute predicts that demand for deep analytical talent could be 50-60% greater than its supply by 2018 and that there will be a shortage of approximately 1.5 million data analysts and data saavy managers [71]. Given that data analysts are a scarce resource and the magnitude of potential value they can add to various industries, it is critically important to increase the scale and efficiency at which professional data analysts can work, and to enable a broader audience to participate in analysis.

In recent years, much work has resulted in improvements in visual analytic software. For instance, tools such as Tableau [100] enable analysts to express and evaluate the results

of database queries without writing complex SQL queries. Such tools permit analysts to query, explore and present data more rapidly.

In practice, however, data sets often contain data quality issues that impede effective use of these tools. For instance, data is often stored in idiosyncratic formats incompatible with downstream analysis and visualization tools. Other anomalies such as missing, erroneous, extreme, or duplicate data can undermine analysis and are time-consuming to find and fix.

As a result, analysts and domain experts spend an inordinate amount of time diagnosing data quality issues and reformatting data before performing meaningful analysis. Some estimate that such data cleaning constitutes 50-80% [19] of the development time and cost in data warehousing projects. Data wrangling often involves writing scripts in languages such as Python or R, or using expensive ETL tools. In some cases, less technical audiences are unable to work with their data.

The database and statistics communities have developed numerous automated routines for data cleaning. These automated routines can help identify potential anomalies in a data set. However, purely automated data cleaning currently suffers from limitations. First, such routines are rarely 100% accurate. For instance, entity resolution algorithms can identify potential duplicates in a data set. However, an analyst must often assess whether multiple records are in fact duplicates. Second, even when routines can accurately identify potential anomalies, human judgement is often necessary to choose appropriate transformations to fix them. As a simple example, it is trivial to detect missing values in a data set, but it is not immediately clear how to respond to them. Should an analyst delete the values or use one of a myriad of common imputation methods [2]?

Interactive visual tools have been introduced for data cleaning tasks such as schema matching, entity resolution, and data reformatting. However, most systems for working with data are non-interactive and inaccessible to a general audience, while those that are interactive make only limited use of visualization and direct manipulation techniques. On the other hand, most visualization research assumes that input data arrives pristine, too often turning a blind eye to concerns of data formatting and quality. This disconnect suggests an opportunity to combine automated techniques with interactive visual interfaces to support data wrangling.

In this thesis, we focus on the problem of designing interfaces to support the iterative

process of data assessment and transformation. We approach this problem by coupling automated routines with interactive interfaces. The use of interactive interfaces helps alleviate the verification and decision making problems of purely automated routines. Automation can improve efficiency and also help end-users specify otherwise difficult transformations. Moreover, in some cases interaction can provide feedback to automated routines, improving their accuracy.

1.1 Contributions

This thesis contributes novel methods and systems for integrating automated routines with interaction and visualization techniques to improve the efficiency and scale at which data analysts can assess and transform data. In particular, we make contributions in three categories:

Characterization of analysis practices and outstanding challenges.
Through an interview study with 35 professional data analysts, we characterize the process of industrial data analysis and document how organizational features of an enterprise impact it. Our characterization includes **five high-level tasks**: *discovery, wrangling, profiling, modeling* and *reporting*. We describe recurring pain points, outstanding challenges, design implications and barriers to adoption for visual analytic tools. Example pain points include discovering relationships in large warehouses, transforming data sets into usable formats, assessing data quality issues and operationalizing workflows. We find that visual analytic tools typically lack support for the *wrangling* and *profiling* stages. This thesis primarily addresses these stages.

Novel interactive systems for data wrangling.
We introduce Wrangler, a graphical interface for data transformation. Wrangler combines direct manipulation of visualized data with automatic inference of relevant transforms, enabling analysts to iteratively explore the space of applicable operations and preview their effects. We present a user study results showing that Wrangler significantly reduces specification time and promotes the use of robust, auditable transforms instead of manual editing.

We then present Profiler, a visual analytic tool for assessing data quality issues. Profiler's architecture combines extensible data types, anomaly detection routines and summary visualizations. The system contributes novel methods for integrated statistical and visual analysis, automatic view suggestion, and scalable visual summaries that support real-time interaction entirely in the browser with millions of data points.

Algorithmic support for guiding interactive data wrangling. We present new algorithms for inferring transformations from user interactions, detecting exceptions in data produced by data transformation scripts run at scale, and automatically suggesting coordinated multiple view displays to aid data quality assessment.

1.2 Outline

We begin in Chapter 2 by discussing the results of our interview study. Here, we describe recurring challenges in the analytic pipeline. In the remainder of the thesis, we present systems and techniques for addressing challenges within two stages of the pipeline: wrangling and profiling. In Chapter 3 we review prior work on algorithmic and statistical approaches to data cleaning, scalable interaction and visualization, and existing interactive systems for data transformation. In Chapter 4 we present Wrangler, an interactive system for enabling graphical specification of data transformation script. Next, in Chapter 5, we introduce methods for debugging scripts produced by Wrangler. In Chapter 6, we discuss Profiler, a system that integrates statistics and visualization for assessing data quality. Finally, in Chapter 7, we summarize this thesis and discuss outstanding challenges for building interfaces to enhance analysts' productivity.

Chapter 2

Interview Study of Enterprise Data Analysts

Organizations rely on data analysts to model customer engagement, streamline operations, improve production, inform sales and business decisions, and combat fraud. Though numerous analysis and visualization tools have been built to improve the scale and efficiency at which analysts can work, there has been little research on how analysis takes place within the social and organizational context of companies. To better understand the enterprise analysts' ecosystem, we conducted semi-structured interviews with 35 data analysts from 25 organizations across a variety of sectors, including healthcare, retail, marketing and finance. Based on our interview data, we characterize the process of industrial data analysis and document how organizational features of an enterprise impact it. We describe recurring pain points, outstanding challenges, and barriers to adoption for visual analytic tools. Finally, we discuss design implications and opportunities for visual analysis research.

2.1 Introduction

Organizations gather increasingly large and complex data sets each year. Within organizations, an increasing number of individuals — with varied titles such as "business analyst", "data analyst" and "data scientist" — perform analyses on such data. These analysts constitute an important and rapidly growing user population for analysis and visualization tools.

Enterprise analysts perform their work within the context of a larger organization. Analysts often work as a part of an analysis team or business unit. Little research has observed how existing infrastructure, available data and tools, and administrative and social conventions within an organization impact the analysis process within the enterprise. Understanding how these issues shape analytic workflows can inform the design of future tools.

To better understand the day-to-day practices of enterprise analysts, we conducted semi-structured interviews with 35 analysts from sectors including healthcare, retail, finance, and social networking. We asked analysts to walk us through the typical tasks they perform, the tools they use, the challenges they encounter, and the organizational context in which analysis takes place.

In this chapter, we present the results and analysis of these interviews. We find that our respondents are well-described by **three archetypes** that differ in terms of skill set and typical workflows. We find that these archetypes vary widely in programming proficiency, reliance on information technology (IT) staff and diversity of tasks, and vary less in statistical proficiency. We then discuss how organizational features of an enterprise, such as the relationship between analysts and IT staff or the diversity of data sources, affect the analysis process. We also describe how collaboration takes place between analysts. We find that analysts seldom share resources such as scripts and intermediate data products. In response, we consider possible impediments to sharing and collaboration.

Next we characterize the analysis process described to us by our respondents. Our model includes **five high-level tasks**: *discovery, wrangling, profiling, modeling* and *reporting*. We find that discovery, wrangling and profiling, often the most tedious and time-consuming aspects of an analysis, are underserved by existing visualization and analysis tools. We discuss recurring pain points within each task as well as difficulties in managing workflows across these tasks. Example pain points include integrating data from distributed data sources, visualizing data at scale and operationalizing workflows. These challenges are typically more acute within large organizations with a diverse and distributed set of data sources.

We conclude with a discussion of future trends and the implications of our interviews for future visualization and analysis tools. We argue that future visual analysis tools should

leverage existing infrastructures for data processing to enable scale and limit data migration. One avenue for achieving better interoperability is through systems that specify analysis or data processing operations in a high-level language, enabling retargeting across tools or platforms. We also note that the current lack of reusable workflows could be improved via less intrusive methods for recording data provenance.

2.2 Related Work

Many researchers have studied analysts and their processes within intelligence agencies [13, 43, 59, 60, 83]. This work characterizes intelligence analysts' process, discusses challenges within the process, and describes collaboration among analysts. Although there is much overlap in the high-level analytic process of intelligence and enterprise analysts, enterprise analysts often work on different types of data with different analytic goals and therefore perform different low-level tasks. For example, enterprise analysts more often perform analysis on structured data than on documents and emails.

Others have characterized tasks and challenges within the analysis process [3, 53, 66, 90, 93]. Amar et al. [3] propose a set of low-level analysis tasks based on the activities of students in an Information Visualization class. Their taxonomy largely includes tasks subsumed by our *profile* and *model* tasks and does not address the other tasks we have identified. Russell et al. [90] characterize high-level sensemaking activities necessary for analysis. We instead identify specific tasks performed by enterprise analysts. Sedlmair et al. [93] discuss difficulties evaluating visualization tools in large corporations, including acquiring and integrating data. We discuss common challenges within these subtasks. Kwon and Fisher [66] discuss challenges novices encounter when using visual analytic tools. In contrast, our subjects are largely expert users of their tools.

Fink et al. [24] performed an ethnographic study of cyber-security analysts. They find that visual analytic tools in this domain have limited interoperability with other tools, lack support for performing ad hoc transformations of data, and typically do not scale to the necessary volume and diversity of data. We find similar issues across multiple domains.

Several researchers have articulated the importance of capturing provenance to manage analytic workflows [8, 27, 31, 38]. Such systems often include logs of automatically

recorded interaction histories and manual annotations such as notes. In this chapter, we discuss the difficulty of recording provenance in enterprise workflows, which typically span multiple tools and evolving, distributed databases.

Multiple research projects have demonstrated benefits for collaborative analysis and developed tools to foster such collaboration. Nardi et al. [74] find that collaborating across spreadsheets promotes sharing both programming and domain expertise. Isenberg et al. [52, 53] discuss design considerations for supporting synchronous, co-located collaboration. Similar to intelligence analysts [60], we have found that most enterprise analysts collaborate asynchronously. We discuss how and when these analysts collaborate. Others [18, 37, 107] discuss design considerations to support work parallelization and communication in asynchronous social data analysis. We discuss the types of resources that analysts must share to enable collaboration and the impediments they face.

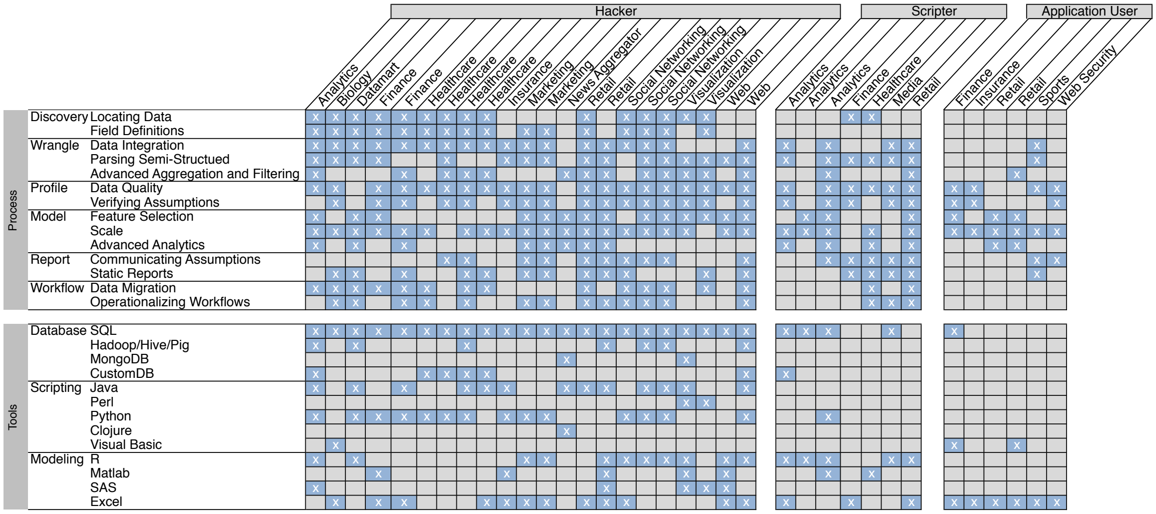
Researchers have also advocated the use of visualization across more phases of the analysis life-cycle [56]. Our analysis corroborates this suggestion. Examples include visualizations to assist schema mapping for data integration [36, 88] and visual analytics for data de-duplication [15, 58]. Our interviews indicate that such tools are sorely needed, and that visualization might be further applied to tasks such as discovery and data manipulation.

2.3 Methods

We conducted semi-structured interviews with enterprise analysts to better understand their process and needs. We use the term "analyst" to refer to anyone whose primary job function includes working with data to answer questions that inform business decisions.

2.3.1 Participants

We interviewed 35 analysts (26 male / 9 female) from 25 organizations. Our interviewees held a number of job titles, including "data analyst", "data scientist", "software engineer", "consultant", and "chief technical officer". The organizations were from 15 sectors including healthcare, retail, social networking, finance, media, marketing, and insurance (see Figure 2.1 for the complete list). The organizations ranged in size from startups with fewer



			Hacker																					
			Analytics	Biology	Datamart	Finance	Finance	Healthcare	Healthcare	Healthcare	Healthcare	Insurance	Marketing	Marketing	News Aggregator	Retail	Retail	Social Networking	Social Networking	Social Networking	Visualization	Visualization	Web	Web
Process	Discovery	Locating Data	x	x	x	x	x	x	x	x	x					x		x	x	x	x	x		
		Field Definitions	x	x	x	x	x	x	x	x	x		x	x		x		x	x	x		x		
	Wrangle	Data Integration	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x				x
		Parsing Semi-Structued	x	x	x	x			x			x	x	x		x	x		x	x	x	x	x	x
		Advanced Aggregation and Filtering	x				x		x	x	x				x	x	x		x	x	x	x		x
	Profile	Data Quality	x	x		x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
		Verifying Assumptions		x		x	x		x	x		x	x	x		x	x	x	x	x	x	x		x
	Model	Feature Selection	x		x	x							x	x	x	x	x		x	x	x	x	x	x
		Scale	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x		x	x
		Advanced Analytics	x		x		x						x	x	x	x	x							
	Report	Communicating Assumptions							x	x			x	x		x	x	x	x	x				x
		Static Reports		x	x		x			x	x		x	x		x	x	x				x		x
	Workflow	Data Migration	x	x	x	x	x	x		x	x					x		x	x	x		x		x
		Operationalizing Workflows		x	x		x	x		x			x	x		x	x	x	x	x				x
Tools	Database	SQL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
		Hadoop/Hive/Pig	x		x					x							x		x	x				x
		MongoDB													x						x			
		CustomDB	x					x	x	x	x													x
	Scripting	Java	x		x		x			x	x	x			x	x	x		x	x	x			x
		Perl																			x	x		
		Python	x		x	x	x	x	x	x		x	x	x				x	x	x				x
		Clojure													x									
		Visual Basic		x																				
	Modeling	R	x		x								x	x			x	x	x	x	x		x	x
		Matlab				x						x					x				x		x	
		SAS	x														x				x	x	x	
		Excel		x		x	x				x	x	x	x		x	x	x					x	x

			Scripter							Application User					
			Analytics	Analytics	Analytics	Finance	Healthcare	Media	Retail	Finance	Insurance	Retail	Retail	Sports	Web Security
Process	Discovery	Locating Data				x	x								
		Field Definitions													
	Wrangle	Data Integration	x		x			x	x					x	
		Parsing Semi-Structued	x		x	x	x	x	x					x	
		Advanced Aggregation and Filtering			x				x				x		
	Profile	Data Quality	x		x	x	x	x	x	x	x			x	x
		Verifying Assumptions	x		x	x		x	x	x	x				x
	Model	Feature Selection		x	x				x	x		x	x		
		Scale	x	x	x		x		x	x	x	x	x	x	x
		Advanced Analytics	x		x		x		x			x	x		
	Report	Communicating Assumptions			x	x	x	x	x					x	x
		Static Reports				x	x	x	x					x	
	Workflow	Data Migration					x		x						
		Operationalizing Workflows					x	x	x						
Tools	Database	SQL	x	x	x			x		x					
		Hadoop/Hive/Pig													
		MongoDB													
		CustomDB	x												
	Scripting	Java													
		Perl													
		Python			x										
		Clojure													
		Visual Basic								x			x		
	Modeling	R	x	x	x			x	x						
		Matlab			x		x								
		SAS													
		Excel	x			x			x	x	x	x	x	x	x

Figure 2.1: Respondents, Challenges and Tools. The matrix displays interviewees (grouped by archetype and sector) and their corresponding challenges and tools. *Hackers* faced the most diverse set of challenges, corresponding to the diversity of their workflows and toolset. *Application users* and *scripters* typically relied on the IT team to perform certain tasks and therefore did not perceive them as challenges.

than 10 employees to corporations with tens of thousands of employees. The analysts ranged from Ph.D. graduates in their first year of work to Chief Data Scientists with 10-20 years of experience.

We recruited interviewees by emailing contacts at organizations within our personal and professional networks. In some cases, we emailed analysts directly. In others, we emailed individuals who forwarded us to analysts within their organization. This recruitment strategy introduces potential bias in our results. For example, the majority of our interviewees were based in Northern California. Also, many of the analysts were sophisticated programmers. To be clear, our research goal is to characterize the space of analytic workflows and challenges, *not* to quantify the prevalence of any specific activity. Other methods, such as surveys or analyzing online job postings, would be better suited for quantifying our findings.

2.3.2 Interviews

We conducted semi-structured interviews with 1 to 4 analysts at a time. We began each interview with a quick introduction describing the purpose of the interview: to understand analysts' day-to-day work practices and any challenges they face. Each interview lasted from 45 minutes to 2 hours. Whenever possible, we interviewed analysts on location at their job. For interviewees outside of Northern California, we conducted interviews over the phone or via Skype.

We asked open-ended questions and encouraged interviewees to describe their lived experiences, such as *"walk us through a recent analysis task"* or *"describe a time you worked with another analyst."* In each interview, we sought to learn the following:

- What tasks do analysts perform?
- What kinds of data sources and formats do they work with?
- What tools do they regularly use and how do they use them?
- How do analysts vary in terms of programming proficiency?
- How do analysts vary in terms of statistical proficiency?
- What are the "results" of analysis?
- What happens to these results "downstream"?
- What are recurring bottlenecks and pain points?
- How important is scalability?
- How important is sociability?
- What is the relationship between analysts and other business units?
- Where are analysts situated within their corporate hierarchy?

In addition to open-ended questions, we asked interviewees to describe the tools and data sets they use within their current organization. During the interviews we took extensive notes.

2.3.3 Analysis

We analyzed our interview data using an iterative coding method. We grouped common practices, tools, challenges and organizational issues into high level categories. We iterated and refined these categories as we gathered more data. In the remainder of the chapter, we describe the types of analysts we encountered and the social context in which they perform analysis. We then describe recurring patterns in the analytic process and enumerate the most common and severe challenges faced. Throughout, we use representative quotes from respondents to support our analytic claims.

2.4 Analysts and Organizational Context

We found that analysts vary widely in their skill sets and common tasks. We categorized analysts into three archetypes based on the tasks they perform and the tools they use. We then report recurring themes in how organizations structured both personnel and data, and discuss collaboration practices within analysis teams.

2.4.1 Analyst Archetypes

We asked analysts to describe recent tasks, the tools they used to complete them, and others in their organization who helped them. Based on the responses, we found that analysts varied widely in their programming proficiency and diversity of tools used. We found that analysts generally fall into three archetypes: *hacker*, *scripter*, and *application user*. For each group we discuss the discrepancies in programming proficiency, the range of tasks typically performed, reliance on IT staff and statistical sophistication.

Hackers

Hackers were the most proficient programmers of the three groups and the most comfortable manipulating data. They typically used at least three different types of programming languages. In addition to working with an analysis package (e.g., R or Matlab), they frequently used a scripting language (Python, Perl) and a data processing language (SQL, Pig [80], *etc*). As one data scientist described:

> *I'm not a DBA, but I'm good at SQL. I'm not a programmer but am good at programming. I'm not a statistician but I am good at applying statistical techniques.*

Hackers typically had the most diverse and complex workflows of the three archetypes, characterized by chaining together scripts from different languages that operate on data from distributed sources. Because of their skill set, hackers often completed flexible workloads without assistance from coworkers such as IT staff. For instance, they were more likely to acquire a new data source outside of the organization's data warehouse and integrate it with internal data. Because of their knowledge of query languages such as SQL or Pig, they could also typically run jobs at scale on their own. In some cases, these analysts even helped build and maintain an organization's central data warehouse and database engines.

Analysts in this group often perform less sophisticated statistical models than scripters. They reported that working with larger data sets limited the types of statistical routines they could run on the data. Also, because this group relied less on IT staff for completing certain tasks, they spent more time in early-stage analytic activities prior to modeling.

Hackers reported using a variety of tools for visualization, including statistical packages, Tableau, Excel, PowerPoint, D3, and Raphäel. Six hackers viewed tools that produce interactive visualizations as reporting tools and not exploratory analytics tools. Since they could not perform flexible data manipulation within visualization tools they only used these tools once they knew what story they wanted to tell with the data.

Scripters

Scripters performed most of their analysis within a software package such as R or Matlab. They were able to perform simple manipulations such as filtering and aggregating data, but typically could not perform custom operations such as parsing log files or scraping data off the web. They generally operated on data that had been pulled from the data warehouse by IT staff and stored in an expected format. Some of these analysts could write simple SQL queries (e.g., without joins) to pull data into their analytic tool of choice. In some cases, they were comfortable writing scripts in a scripting language, but typically do not know

how to create scripts that run at scale.

Scripters applied the most sophisticated models among the analysts we observed. Advanced modeling was potentially enabled by the breadth of libraries available for analytic packages and the percentage of time these analysts devoted to modeling. The implementation and application of algorithms was more easily done when dealing with data resident on one machine (as opposed to distributed). Scripters often produced visualizations using the statistical package during exploratory analysis. Using the same tool for visualization and analysis permitted them to iterate fluidly between the two tasks. In some cases scripters used a separate tool, such as Tableau, to create interactive dashboards for reporting after the significant insights had been discovered.

Application User

The last set of analysts performed almost all operations in a spreadsheet or other dedicated analysis application (e.g., SAS/JMP, SPSS, *etc*). Like scripters, they typically required someone to prepare data for them by pulling it from the warehouse. One Excel user's account was quite typical of most spreadsheet users:

> *All data is in a relational database. When I get it, it's out of the database and into an Excel format that I can start pivoting. I ask the IT team to pull it.*

Application users typically worked on smaller data sets than the other groups and generally did not export data from the spreadsheet except for building reports. In some cases, advanced application users wrote scripts using an embedded language such as Visual Basic. To produce visualizations they typically created charts in Excel or exported data to a reporting tool such as Crystal Reports.

2.4.2 Organization

Enterprise analysts work within the context of a larger organization. Political and social conventions within the organization can and do affect the analysis process. We now discuss three recurring themes.

The Relationship between Analysts and IT Staff

Analysts often interacted closely with IT staff to complete aspects of their job. We observed that the IT team regularly provides four primary functions in support of analysis. First, they often maintain data within a centralized warehouse. This maintenance includes ingesting new data sources and ensuring quality across these sources. If an analyst requires new data in the warehouse, the analyst will often work with IT staff to communicate these requirements.

Second, the IT team assists analysts in acquiring data. Analysts, especially application users and scripters, rely on IT staff to query data from the warehouse and export it in an accessible format. For instance, 12 analysts reported having the IT team write SQL queries and convert the output to delimited files or spreadsheets.

Third, the IT team is responsible for operationalizing recurring workflows. One analyst at a media company described the workflows he built as "prototypes". After experimenting with samples of data, the analyst would send a high-level description of the workflow steps — written in English — to IT staff. IT staff then implemented the process to run reliably and at scale. Even hackers relied on IT staff to operationalize tasks that were critical to other business units or had challenging scalability requirements. For example, one analyst at a hedge fund required the IT team to operationalize his workflows to achieve low-latency for high-frequency trading.

Finally, the IT team serves as a source of documentation. Even analysts comfortable writing complex SQL and Hadoop jobs often require IT staff to help find the appropriate data and understand its structure and schema. This reliance on IT staff was particularly true in organizations where data was distributed across many different data sources. Hackers were most likely to use the IT team explicitly for this function, as they were more likely to access data directly from the warehouse. Scripters and application users relied on this function implicitly when receiving data from members of IT.

Distributed Data

Analysts, especially in large organizations, worked with data stored in a variety of repositories. 21 analysts reported working with data stored in at least three different formats.

For instance, three hackers used data stored in spreadsheets in a shared file system, account data stored in a relational database, and log files stored in HDFS (the Hadoop distributed file system). Many analysis tasks involved integrating data from multiple sources:

> *We had three different sources of data for a customer support interaction. Data gets tracked in customer support, part of Salesforce. The end-user data is stored in our user management system. And all the events are logged into the data warehouse event logs. And if a user buys, this gets logged in the credit card system. So you may have four sources. For me, I have to pull data out of each data source. As it turns out, all the data is in different warehouses ...in different schemas.*

Some analysts performed this integration themselves. In other cases, analysts — especially application users and scripters — relied on the IT team to assemble this data for them.

Consumers of Analysis

The results of analysis served many different departments within the organization. For instance, analysts worked with marketing, business development, sales, operations, and design teams. They often translated high-level business questions into low-level analytic tasks. At the end of analysis they typically generated reports in the form of summary statistics, charts or recommendations. Other analysts also consumed these reports to inform future analysis.

Analysts typically shared static reports in the form of template documents or PowerPoint presentations. In some cases, the results of analysis were dynamic reports such as interactive dashboards that enabled end users to filter or aggregate data. In other cases, reports were simply recommendations of actions to take. The results were typically shared via email, a shared file system, or during a group meeting or presentation. Consumers and analysts often iterated on reports to pursue newly-generated hypotheses, modify assumptions, or redefine the problem specification. Because consumers often translate their questions loosely, analysts sometimes misinterpret them. After arriving at a result, these results were often archived in a personal or shared drive, but were not consulted on a regular basis and were difficult to search:

> *We meet every week, we find some interesting insight, and we say that's great. There's no repeatable knowledge, so we end up repeating the process 1 year later.*

2.4.3 Collaboration

In addition to working with IT staff and other business units, analysts were often part of their own analysis unit. Here we describe at which points in their process analysts collaborated and which resources they shared throughout their workflows.

Collaboration Process

Analysts reported meeting regularly with other analysts to discuss long-term projects and immediate next steps. However, most analysts reported that they rarely interacted with other analysts to complete a given task. One sentiment, echoed by many analysts, was *"working on a team is the exception in my experience."*

Shared Resources

We found analysts shared four types of resources: data, scripts, results and documentation. All the organizations we talked to contain some central repository through which analysts access a large proportion of their data. During analysis, it was common to perform transformations such as sampling, filtering or aggregating data to compute intermediate data products that were used for downstream analysis. These products were typically disseminated via email or on shared drives. In some cases, these intermediate products were stored in the data warehouse. In a few organizations, there were attempts to monitor when new data sets were created. One analyst described a chat room that all analysts monitored throughout the day. When someone created a new data set, a message was *"automatically sent to the chat room, producing ambient knowledge."*

The least commonly shared resource was data processing scripts. We found that analysts typically did not share scripts with each other. Scripts that were shared were disseminated similarly to intermediate data: either through shared drives or email. Analysts rarely stored their analytic code in source control. Analysts with engineering backgrounds

noted the difference in process between product code and analysis code; one joked that even when she stored code in a repository, *"svn is more like backup than version control."*

On the other hand, analysts frequently shared their results with each other. These results often took the form of reports or charts. Some analysts used Crystal Reports, others constructed graphs or simply presented summary statistics of their models. Analysts viewed these results during planning meetings or presentations, but did not typically consult these reports during subsequent analysis. The results were not stored in a searchable repository. Most reports were static, preventing others from modifying input parameters or assumptions to see how the results might change. In a few cases, the result of analysis was a parametrizable function or an interactive dashboard with support for filtering or aggregating data.

Impediments to Collaboration

We observed three common impediments to collaboration. First, the diversity of tools and programming languages made it difficult to share intermediate code. One quantitative analyst at a hedge fund reported that sharing scripts was too difficult because of the diversity of languages the analysts used: one analyst used Mathematica, another used Perl, and he used Python.

Second, the same analysts reported that finding a script or intermediate data product someone else produced was often more time-consuming than writing the script from scratch. Many (18/35) reported that it was difficult to search for a desired intermediate product. These products were often difficult to interpret, as documentation for these data sets was often more sparse than for data in the warehouse. Because of this, analysts resorted to "blast emails" such as *"has anyone made a data set filtering out users from Latin America?"* This difficulty may in part result from the way data and scripts were typically shared: by storing them on a shared drive or via email. Even when an analyst could find an appropriate script or data set, the product may lack documentation of how it should be used and what assumptions must hold. As one analyst said:

> *You're wary of reusing code because if you blindly reuse it you might miss obvious things that are important to my own code... the same data set can be*

> *used for thousands of analyses, so unless you are reusing it for the same exact thing then you might make mistakes.*

Many analysts (25/35) also expressed a general attitude that intermediate products such as code and data were *"ad hoc"*, *"experimental"* or *"throw-away."* As a result, analysts spent less time writing modular, reusable, and parametrizable code and rarely produced documentation. Other analysts noted that a lot of their work ultimately did not validate any useful hypothesis, and so they end up discarding the data or code. One analyst observed, *"you go down a lot of dead ends, and you come up with a bunch of hypotheses. 8 out of 10 are dead ends."* The same analyst went on to say he lacked a process to tell others *"don't look for the correlation here, because I looked and its not here. Especially your dead ends – there are no remnants of that."* Analysts intentionally discarded intermediate products because the end result did not seem to be insightful.

2.5 Challenges in the Analysis Process

We characterized the tasks within the analysis process based on interviewees' descriptions of their work. We identified five high-level tasks that repeatedly occurred in respondents' analysis efforts:

Discover data necessary to complete an analysis tasks. Example tasks include finding a data set online, locating the database tables in a MySQL database, or asking a colleague for a spreadsheet.

Wrangle data into a desired format. Example tasks include parsing fields from log files, integrating data from multiple sources into a single file or extracting entities from documents.

Profile data to verify its quality and its suitability for the analysis tasks. Example tasks include inspecting data for outliers or errors and examining the distributions of values within fields.

Model data for summarization or prediction. Examples include computing summary statistics, running regression models, or performing clustering and classification.

Report procedures and insights to consumers of the analysis.

Not all analyses require all five tasks, and not all analysts perform each of them. We now discuss recurring pain points and the challenges of managing workflows across tasks.

2.5.1 Discovery

Throughout their work, analysts acquired data necessary to complete their tasks. Within large organizations, finding and understanding relevant data was often a significant bottleneck.

Where is my data?

For 17 analysts, finding relevant data distributed across multiple databases, database tables and/or files was very time consuming. Organizations often lacked sufficient documentation or search capabilities to enable efficient identification of desired data. Instead, analysts relied on their colleagues: they often asked database administrators or others for help. One analyst described the problem:

> *It is really hard to know where the data is. We have all the data, but there is no huge schema where we can say this data is here and this variable is there. It may be written but the wiki is very stale: pointers don't point to the right place and it changes really fast. The best thing you can learn working here is whom to ask, because in their head a lot of people know a lot of stuff. It's more like folklore. Knowledge is transmitted as you join.*

Some organizations also restricted access to data, requiring an appropriate administrator to grant privileges. In some cases, the administrator who set up the data may have already left the company.

Field Definitions

The difficulties in discovering data were compounded by the difficulty of interpreting certain fields in a database. In at least 16 instances, analysts described situations in which fields were coded and required lookups against external tables. Foreign key definitions help identify the appropriate table to perform lookups, but these definitions were often

missing in relational databases and non-existent in other types of data stores. Even without coding, missing units or metadata created ambiguity. For instance, one analyst noted that many date-time fields were stored without timezone information. The analysts had to reconstruct timezones from corresponding geographic information.

In 8 reported cases, schema drift lead to redundant columns. One company we interviewed had a database table containing four columns containing job titles for its users. These columns evolved over time, were often conflicting and there was no documentation describing which column was up-to-date or appropriate to use.

2.5.2 Wrangling

Once an analyst discovered the appropriate data to use, she often needed to manipulate the acquired data before she could use it for downstream analysis. Such data wrangling, munging, or cleaning [56] involves parsing text files, manipulating data layout and integrating multiple data sources. This process, whether managed by IT staff or by analysts, was often time consuming and tedious.

> *I spend more than half of my time integrating, cleansing and transforming data without doing any actual analysis. Most of the time I'm lucky if I get to do any analysis. Most of the time once you transform the data you just do an average... the insights can be scarily obvious. It's fun when you get to do something somewhat analytical.*

Prior work [23, 85] lists a number of data wrangling challenges. We identify three common issues shared by our interviewees.

Ingesting Semi-Structured Data

A number of analysts (23/35) reported issues processing semi-structured data. The most common example was ingesting log files. Parsing log files may require writing complex regular expressions to extract data into relevant fields. Interleaved log files containing multiple event types in a single file can further complicate parsing.

One analyst described working on log files generated by a tele-communications company. The log files contained three types of records for text messages: outbound messages, inbound messages, and receipt confirmation. The analyst needed to define criteria to divide the logs into logical segments. Verifying that criteria accurately splits records can be difficult, especially with collections of log files containing terabytes of data.

So-called "block data" is another common data format that was difficult to parse. In a block format, logical records of data are spread across multiple lines of a file. Typically one line (the "header") contains metadata about the record, such as how many of the subsequent lines (the "payload") belong to the record.

Data from 3rd party services often required a level of processing before analysis could begin. Email campaign providers, credit card processing companies, and other external services often delivered user reports in idiosyncratic formats. One analyst responded that:

> *"[Email campaign providers are] very inconsistent about feed format. Bounces, responses, etc ... are hard to bring in in the right format. We are constantly munging the data to get into our SQL database."*

Although many data sets arrived in these formats, most visualization tools do not support such semi-structured formats, preventing their use at early stages of data transformation.

Data Integration

Another difficulty, reported by 23 analysts, was integrating data from multiple sources. Identifiers useful for joining records across data sets were often missing in one or more data sets, inconsistent across data sources or incorrect for certain records. One hospital analyst recounted issues integrating patient medical data:

> *The easiest patient identifier is the medical record number (MRN). We should have consistent MRNs in any data source but 5 to 10 percent of MRNs are mistyped or incorrect or blank. In emergencies, a patient may get assigned a temporary MRN. Later it's reassigned but sometimes we forget to reassign. We have to identify patients by other means: first name, last name, birthdate,*

> *gender. Several data points together might identify a single patient. There may be slight inconsistencies.*

When identifiers were missing or incorrect, analysts derived new methods for linking records. Like the medical researcher above, analysts would match records using rules based on other fields that did not uniquely identify distinct records.

Analysts reported three types of inconsistency in identifiers during integration. First, identifiers used slight variations in spelling or formatting that make direct matches difficult. For instance, a patient's first name might be stored as "John" in one record and "Jonathan" in another. Some analysts defined ad hoc rules ("fuzzy matches") to detect similar items. The analysts then inspected the matches to verify that the records referred to the same entity.

Second, data sources used two different encodings to represent the same identifier. For instance, a state might be identified by its full name (e.g., California) or by its as Federal Information Processing Standard (FIPS) code (e.g., 6). In this case, an analyst must find or construct a mapping between identifiers.

In the third case, identifiers used inconsistent units of measurement or class definitions. Multiple analysts described attempts to consolidate their respective company's industry codes with the North American Industry Classification System (NAICS). Others described difficulties integrating geographic data with varied region definitions. Similarly, many data sets use overlapping conventions for financial quarters. The situation is complicated when sets of regions overlap and one standardization does not subsume the others:

> *The biggest challenges have been making two sources work together in terms of units of analysis. Our industry titles are different than standard industry titles... it would be nice to have mappings between standardizations. We are matching internal standards and outside standards which is hard... We have a region "SF Bay Area" which includes San Jose. CVSA is mapped to metro areas and so San Fran and San Jose are different areas. We need a method of grouping, but they won't even overlap the same. It's worse than hierarchical. You end up losing the data somewhere along the path.*

These integration problems were made more difficult when the data was stored across multiple databases. In response, most analysts reported having to migrate all of the data sets into the same data processing framework.

The lack of support for data integration also impedes the effective use of exploratory visualization tools. Because analysts were often unable to integrate external data to augment visualizations within a tool, they must resort to assembling data outside of the tool. One analyst noted that she spent most of her time integrating data together from disparate data sources to drive visualizations.

Advanced Aggregation and Filtering

Some analysts (16/35) noted difficulty performing ad hoc grouping of observations, as in path or funnel analysis [99]. One analyst at a web company investigated the sequence of actions users took before converting to a paying customer, upgrading their accounts, or canceling their accounts. The source data set was a log of user activities on the website, with each entry corresponding to a single activity by a single user. The analysts needed to group activities not only by user, but also by event time, where the time was conditional on other events in the log (i.e., prior to closest conversion). These types of queries in SQL often involve nested subqueries. Similar subqueries are necessary to write filters such as "delete all users who never upgraded their account." Analysts find it difficult or impossible to express these queries within current visualization tools. In response, they must process data outside of the tool and then load it back in, significantly slowing their analysis process.

2.5.3 Profiling

Once required data is assembled and integrated, analysts enter a phase of diagnosing data quality issues and understanding what assumptions they can make about their data. As one analyst quipped: *"I'd rather the data go away than be wrong and not know."*

Data Quality

Data sets may contain a number of quality issues that affect the validity of results, such as missing, erroneous or extreme values. Many analysts (22/35) reported issues dealing with

missing data. In some cases, observations contained missing or null attributes. Analysts reported using a number of methods for imputing missing values. One organization even had an intern build a dedicated interface for displaying and correcting missing values across data sources. In other cases, entire observations were missing from a data set. Missing observations were much more difficult to detect.

Another common problem is heterogeneous data in a column: a column with an expected type may contain values of another type. This might occur due to errors in automated processes such as log file generation, errors in human data entry, or because of an explicit decision to "overload" the use of a column. For example, a database table at one organization contained a longitude field that was empty for many of the observations. Instead of creating a new column, some analysts decided to overload this field to store additional data unrelated to longitude. This type of error also occurred when IT teams introduced a new field into a log file, breaking existing scripts that expect the files in a certain format.

Other errors include multivariate constraints. For instance, one analyst described a scenario:

> *in one data set there were 4 males [between the ages] 0 to 9 who were pregnant. If I make an assumption about what that means and filter the data, then I am destroying data. For instance, you might infer hospital care in that particular hospital is not very good.*

Analysts reported using visualization and statistical routines to detect errors in their data. One medical analyst described using visualization for outlier detection in machine-generated data: *"We don't have probability rules to detect outlying events. Once we look at enough data, we'll know exactly what is an artifact."* Others relied more heavily on statistical methods to detect outliers: *"I find [visualization] pretty useless. I'm very much a numbers hound. I'm more into playing with the data. I'm very comfortable with numbers. Visualization adds a layer between me and numbers."* This analyst inspected distributions and summary statistics and identified observations that fell outside the normal range. Generally, most analysts reported using a combination of visualization and statistics to inspect data. During inspection, they were also able to gain an understanding of what assumptions they could make about their data.

Assumptions

Analysts make assumptions during analysis that inform the types of transformations they use, how they sample data and which models are appropriate. Common assumptions included how values were distributed within an attribute (was an attribute normally distributed?), what values were unique (were there duplicates?), and how different attributes related to each other (was X always greater than Y?). Other assumptions required domain expertise to verify.

> *Once you play with the data you realize you made an assumption that is completely wrong. It's really useful, it's not just a waste of time, even though you may be banging your head.*

An analyst in online advertising described an analysis of ad impressions. For a given campaign, the analysis assumed there were at most 15 impressions per user. However, they saw that some users were receiving up to 300 impressions. The analysts then

> *checked if the campaign settings were set correctly, talked to other people about the logic, and then finally started zeroing in on those people. Then you realize if they change states then they are eligible for another 15 [impressions]. Then it affects how you organize the campaign. In practice it tends not to be just data prep, you are learning about the data at the same time, you are learning about what assumptions you can make.*

2.5.4 Modeling

After all the required data was assembled and understood, analysts could begin modeling their data.

Feature Selection

Many analysts (20/35) reported the biggest difficulty in constructing a model was understanding which of the data fields were most relevant to a given analysis task. It was

particularly difficult to understand the relationships among features spread across multiple databases. Also, many fields needed to be transformed before useful patterns would emerge. As one analyst said:

> *In practice right now the biggest differentiator is feature selection: knowing what columns to pay attention to and how to sensibly transform them. Do you take the log of these, do you combine these two? A lot of work is just finding what the units of the columns should be.*

Scale

Most respondents (31/35) noted that existing analytic packages, tools or algorithms did not scale with the size of their data sets. The threshold for when data sets were too big was obviously different depending on the tool used. For instance, some application users still used Microsoft Excel 2007, because their organizations would not allow them to upgrade to newer versions. In these cases, analysts could not perform analysis on more than 1,000,000 rows. Scripters were typically limited by the memory requirements of their machine.

Hackers were less limited by large amounts of data, because they could typically run distributed jobs over multiple machines. However, hackers were often limited by the types of analysis they could run because useful models or algorithms did not have available parallelized implementations. As one hacker described, it is difficult to *"take powerful algorithms that work on medium data and make them pluggable in the big data stack."*

Other analysts used sampling but cited that sampling was hard to do correctly without introducing bias into the analysis. Some noted that sampling was especially difficult when the "interesting" observations were sparse. One analyst described their difficulty performing sampling for modeling conversions during funnel analysis:

> *Subsampling can exclude information you actually need... It's not very reasonable for infrequent observations. If you sample down you lose a lot of conversions.*

Issues with scale were even more prominent when dealing with visualization tools. In two cases, respondents had not heard of existing tools (such as Tableau) that would have

been sufficient for their reported data sizes. For others, scale was fundamentally an issue, both in terms of the number of observations and the number of attributes. Existing visualization tools simply could not load enough data into the tool. In other cases, data could be loaded, but operations such as brushing and linking could not be performed at interactive rates. To cope with scale, three of our respondents were building custom data processing engines. One company built their own database engine that pre-computes possible combinations of filters and rollups in their charts. To combat combinatorial explosion, they analyze which columns are typically viewed together.

Interviewees also believed that visualization does not scale to high dimensional data. Some stated that most exploratory tools do not allow them to visualize more than two or three dimensions:

> *[G]raphical representation is at best two or three dimensional. Three dimensions won't tell me very much about how 300 variables interact.*

Visualizing Statistical Models

Analysts would like to apply advanced analytics routines and visualize the results. Though many tools have facilities such as drawing best-fit regression lines, analysts using more advanced machine learning methods (14/35) expressed a desire for visualization tools to help explore these models and visualize their output. However, analysts' descriptions of these potential tools were often vague and imprecise: they sensed a need, but were unsure of the form that a successful solution would take.

2.5.5 Reporting

Analysts typically reported insights gained from modeling to other analysts or business units. The two most-cited challenges were communicating assumptions and building interactive reports.

Communicating Assumptions

One complaint about distributing and consuming reports (made by 17 analysts) is poor documentation of assumptions made during analysis. Analysts typically performed a sequence

of operations that can affect the interpretation of results, such as correcting outliers, imputing missing data or aggregating data. These operations are often context specific, with no standards for each analysis.

In other cases, analysts imposed their own definitions on under-specified concepts. One medical analyst analyzed patient episodes that correspond to all visits to a hospital to treat a given symptom. However, the database did not contain an episode identifier associated with each patient visit. The analysts had to use heuristics, such as the duration between visits, to group hospital visits into episodes. This heuristic was imprecise, as hospitals may treat a patient concurrently for two different symptoms or for the same symptom after a long period of time. Analysts often lost track of all the operations they performed and their rationale for performing them.

Even when assumptions were tracked, they were often treated as footnotes instead of first-class results. One analyst cited that his boss often looked at summary charts without reading the fine print. For instance, an average calculated from three data points would be marked with an asterisk that was then regularly overlooked.

Static Reports

A number of analysts (17/35) also complained that reports were too inflexible and did not allow interactive verification or sensitivity analysis. Often reporting and charting tools were used directly on the output data and contained no knowledge of how the original input data was filtered, transformed or modeled. Much of this work was done before output data was loaded into the tool. Because reporting tools have no access to data provenance, it was often impossible to modify parameters or assumptions to see how the conclusions would change. Viewers can not verify questions such as “how might user acquisition rates change if more money was spent on marketing?”

2.5.6 Workflow

We found that analysts engaged in an iterative, non-linear process in which they cycle among the tasks described above. Managing workflows across these steps brings a number of its own challenges.

Data Migration

Analysts, especially hackers, often used multiple tools and databases to complete their tasks. Different tools and environments often required data in different formats. About half of our respondents (16/35) claimed that the most tedious part of analysis was moving data between tools and warehouses. One data scientist noted the tedium of having to *"Run a Hadoop job, then run a Hadoop job on results, then awk it... Hadoop job chained to Hadoop job chained to a Python script to actually process data."* Scripters and applications users often used separate tools for reporting than they used for wrangling, profiling and modeling.

Operationalizing Workflows

During analysis, analysts generated a number of intermediate products including scripts, spreadsheet formulas and data sets. It was often difficult to assemble these products into a repeatable, reliable and scalable process. Analysts reported that they often explored multiple hypotheses in parallel and create multiple intermediate data products in the process. Reconstructing a repeatable workflow is difficult without a coherent linear history of the operations performed. Even with a coherent history, an existing workflow may break when applied to new or updated input data. This new input data may contain nuances not accounted for that would cause existing code to break. Finally, analysts reported that they wrote experimental code that could not run on large data sets or at necessary speed in real-time systems. They therefore required the IT team to operationalize many of their workflows.

2.6 Future Trends

Looking forward, trends in technology and the analytic workforce will compound the challenges faced by enterprise analysts, with strong implications for the design of visual analytic tools.

2.6.1 The Availability of Public Data

As more and more public data — including government records, financial records, and social network activity — becomes available, organizations will allocate more resources to ingest and integrate this data with their own. Ingesting publicly available data can often be difficult, requiring analysts to crawl and scrape websites or parse data from unstructured and semi-structured sources. In some cases, public data is made accessible through web APIs. In many other cases, organizations — especially those required by law to disclose information — release data in formats that are difficult to process (such as PDF files). An analyst at a large hedge fund noted that their organization's ability to make use of publicly-available but poorly-structured data was their primary advantage over competitors.

In recent years, there have been an increasing number of so-called "data marts", such as InfoChimps.org, that aim to make public data more accessible. Even so, integrating public data with an internal warehouse poses challenges. As discussed previously, many organizations develop internal coding standards for entities such as geographic locations or industry codes. Often, these codes differ from external data. Two sources of public data might also have different coding standards. Moreover, public data often lacks documentation, posing additional challenges to discovery and profiling.

2.6.2 The Rise of Hadoop

Of our analysts, 8/35 reported using Hadoop and IDC predicts the market for Hadoop software will increase by an order of magnitude by 2018 [78]. The increasing popularity of Hadoop could compound challenges in discovery. With relational databases, organizations typically design a database schema and structure incoming data upon load. This process is often time-consuming and difficult, especially with large complex data sets. With Hadoop, analysts typically take advantage of is its ability to operate on less structured data formats. Instead of structuring the data up front during ingest, organizations commonly dump data files into the Hadoop Distributed File System (HDFS) with little documentation. Analysis of this data then requires parsing the data during Map-Reduce jobs or bulk reformatting to load into relational databases. While remaining unstructured, the data may be difficult to search and profile due to the lack of a defined schema. In some cases, the analysts who

originally imported and understood the data may no longer work at the company or may have forgotten important details.

2.6.3 A Growing Demand for "Hacker" Analysts

Over the next few years, we see three factors driving an increasing demand for "hacker"-level analysts. First, constrained IT departments are making it necessary for analysts to be self-serving. When discussing recruitment, one Chief Scientist said *"analysts that can't program are disenfranchised here"*; IT support was prioritized for shipping products, not helping analysts experiment on code.

Second, the increasing scale of data requires many organizations to perform in-database analytics. Analysis software tools such as R and Matlab do not currently scale. Instead, analytic routines are performed within the data warehouse, typically in a shared-nothing parallel database (such as those offered by Aster, Greenplum, or Teradata) or via Map-Reduce or related higher-level languages such as Pig. Analysts therefore need to be adept at both statistical reasoning and writing complex SQL or Map-Reduce code.

Finally, organizations are frequently relying on multiple processing frameworks and tools as requirements evolve. For instance, some organizations will use relational databases to support interactive queries and analysis, rely on Hadoop for batch jobs and processing log files, and also require analysts who can build "prototype" models in R. One analyst noted:

> *Diversity is pretty important. A generalist is more valuable than a specialist. A specialist isn't fluid enough. We look for pretty broad skills and data passion. If you are passionate about it you'll jump into whatever tool you need to. If it's in X, I'll go jump in X.*

These observations are supported by a recent McKinsey report [71] which estimates the demand for big data analysts (a category similar to our observed "hackers") will triple by 2018.

2.6.4 Analysis Teams Are Growing

As the number of analysts increase across organizations, the size of analytic teams should also grow. We expect that efficient collaboration will become both increasingly important and difficult. We see a growing emphasis on better collaboration practice within the larger organizations we observed. This emphasis was shared particularly among managers who observed the inefficiencies of poor collaboration amongst their subordinates. The managers noted that the inefficiencies led not only to repeated work but to inconsistent results. In one large retail company, the directer of six analytic teams noted that multiple analysts would submit conflicting reports of a metric, such as turnover. The analysts used inconsistent assumptions to calculate their results, most of which were not communicated to the business units consuming these reports.

2.7 Design Implications

We now discuss design implications for visual analytic tools based on the challenges and trends identified in our interviews.

2.7.1 Workflow Breakdowns

Our interviews suggest that many of the breakdowns in analysis occur in the early phases of a workflow or transitioning between tasks in a workflow. Despite this, visualization is often typically applied to isolated late stages of the workflow, including reporting and exploring a single data set at a time. Despite much research from the database and statistics communities [12, 23, 41, 45, 84, 85], little visualization research addresses discovery, wrangling or profiling challenges. Visual analytic tools that enable efficient application and assessment of these data mining routines could significantly speed up the analysis process.

Tools that extend their data query facilities to operate over partially structured data will enable analysts to immediately apply visualization and analytics to a much wider set of data and better serve early-stage analysis. Such tools may require additional algorithms and interaction techniques for type induction and structure inference. As an example, multiple analysts cited the popularity of the commercial tool Splunk, which enabled them to

write queries directly against log files without first structuring their data. The analysts noted that Splunk had limited support for visualization and creating dashboards, leaving an opportunity for visualization systems that could enable analysts to begin visualization over unstructured data. Splunk's command-line interface was popular among analysts experienced in programming, but not approachable for those with less experience.

Tools that can connect directly to existing data warehouses can better integrate into analysts' workflows by limiting data migration. If a tool uses its own proprietary data source to process data, then an analyst must migrate data in and out of the tool for it to be useful, impeding fluidity. One analyst liked Google Refine's graphical interface for data transformation, but found it unsuitable for cleaning data in his SQL database because *"that requires me to export everything to CSV and play around there and then I have to put stuff back in the database."*

Analysis is often an iterative process of acquisition, wrangling, profiling and modeling. Although many tools today contain transformation languages, most lack support for common transformation tasks such as integrating new data, window functions and filters or aggregates with nested subclauses. For instance, languages should support filters that remove all employees with salaries in the 95th percentile and window functions that compute rolling averages in time-series data. More complex transformations might be more easily represented with procedural or imperative programming. The lack of support for such transformations requires analysts to transform their data outside of their tool.

Of course, increasing the complexity of a system increases the engineering burden of supporting direct connection to existing data warehouses. Well-designed declarative languages can decrease this burden by limiting the number of primitive operations that need to be implemented across various run times. Still, constraints on data processing languages may make it difficult to run certain types of transformations at scale, or at all, directly within a data warehouse. As one example, transformations that rely on relative positions of observations within a data set are not expressible in standard SQL.

2.7.2 Support Scalable Visual Analytics

One clear implication of our studies is the need for visualization methods that scale. Scaling visualization requires addressing both perceptual and computational limitations. Visualizations that render raw data suffer from over plotting with even moderately large data sets and certainly when applied to data sets containing billions of observations. Visual analytic tools must consider using density or aggregation based plots such as histograms and binned scatter plots [9] for large data sets.

One approach to improved scalability is to leverage existing data processing engines for manipulating data. By creating adapters to common systems such as parallel databases and Hadoop, analytic tools can leverage existing infrastructure to scale to large data sets. For instance, Tableau can translate statements in its internal representation into queries that run on distributed databases. However, simply connecting to existing systems can not achieve interactive rates supporting brushing and linking over large data sets. Visual analytic tools could benefit from server-side pre-processing and aggregation to enable interactive exploration in the client.

Tool builders should also consider how approximation approaches might be applied to scale visual analytic tools. Sampling data can speed up querying but may introduce bias [25]. Ideally, various sampling strategies could be applied directly to a data source from within the tool. This capability would enable more fluid application of various strategies and evaluation of their effectiveness. Other approximation approaches might include online aggregation [25, 42], whereby analysts can visualize the incremental results as they are computed. It remains future work to examine how low-latency query processing over data subsets of various resolutions impact both the quantity and quality of analysis.

2.7.3 Bridge the Gap in Programming Proficiency

The increasing demand for “hackers” highlights the types of tasks that need to be achieved to perform analysis within an enterprise. The inability of scripters and applications users to manipulate data from diverse data sources and at scale makes them dependent on others and limits their effectiveness. Visual analytic tools should strive to bridge this gap in

programming skill by providing direct manipulation interfaces for tasks such as data acquisition and wrangling. To empower hackers, direct manipulation interfaces might also expose the underlying logic of the tool.

2.7.4 Capture Metadata at Natural Annotation Points

If available, a tool should augment intermediate products such as scripts and data with additional metadata. Such metadata might include the script's author, the rationale for an analysis procedure or assumptions made about the input data. The metadata can enable more efficient search over products and simplify the interpretation of results by others. How to best represent and interact with this metadata could itself be an interesting visual analytics problem.

However, many analysts are hesitant to spend time documenting their process because of the number of dead-ends they encounter and intermediate products that get thrown away. One approach to record metadata is to instead increase the utility of recorded metadata by imposing conventions or constraints. Where a user has to make a decision (i.e., file naming), can tools help them make a more useful choice? For instance, many analysts save intermediate data sets in files. All these files will require names, in which analysts often record valuable metadata in an inconsistent and unstructured format; e.g., using "customers_europe_18_24" to indicate they created a file storing customer data for European customers aged 18 to 24. Instead, a tool might impose some structure on the naming procedure so that this metadata can be searched over more easily in the future. By intervening at already existing annotation points, tools might limit the perceived overhead of annotation.

2.8 Conclusion

This chapter presented the results of interviews with 35 data analysts within commercial organizations. We presented a model of phases of analytic activity and enumerated the challenges faced by analysts within these phases. Finally, we discussed the consequences of trends in technology and human resources, and presented corresponding design implications for visual analysis tools.

As the scale and diversity of data sources increases within enterprises, there is an opportunity for visual analytic tools to improve the quality of analysis and the speed at which it takes place. Tools that simplify tasks across the analytic pipeline could empower non-programmers to apply their statistical training and domain expertise to large, diverse data sets. Tools that help manage diverse sets of procedures, data sets, and intermediate data products can enable analysts to work and collaborate more effectively.

In the remainder of this dissertation, we present two tools, Wrangler and Profiler, to address challenges in the wrangling and profiling phases, respectively. We focus on wrangling because it is often the most time-consuming phase of the pipeline and profiling because of the impact an understanding of data quality has on the confidence of a downstream analysis. Also, we hypothesized that both phases would benefit drastically from combining automated techniques with interactive visualization; at the same time both phases are underserved by existing tools.

Chapter 3

Related Work

This thesis draws on and extends related work in interactive interfaces for transforming data, classifying and detecting anomalies in data, and visual analysis systems.

3.1 Data Transformation

Wrangler (Chapter 4) draws on prior work in end-user programming and shares some characteristics with existing data transformation tools. Wrangler leverages programming-by-demonstration techniques to aid specification of transformations in it's underlying declarative language. The system integrates and extends features, such as type inference, of other interactive data cleaning tools. Below, we discuss how Wrangler leverages and augments algorithms and techniques used for interactive data transformation.

3.1.1 Programming-By-Demonstration

Many data cleaning applications apply direct manipulation and programming-by-demonstration (PBD) methods to specific cleaning tasks. Users of SWYN [6] build regular expressions by providing example text selections and can evaluate their effect in visual previews. Potluck [50] applies simultaneous text editing [73] to merge data sources. Karma [103] infers text extractors and transformations for web data from examples entered in a table. Vegemite [69] applies PBD to integrate web data, automates the use of web services, and generates

shareable scripts. Other interfaces [54] apply PBD to data integration via copy and paste actions.

Wrangler applies a number of these techniques: it infers regular expressions from example selections [6] and supports mass editing [50, 73]. Wrangler uses semantic roles akin to Topes [92] and provides natural language descriptions of transforms [69]. However, Wrangler differs in important ways. PBD data tools support text extraction or data integration, but lack operations such as reshaping, aggregation, and missing value imputation. Prior tools (except for Vegemite [69]) also do not generate scripts to document provenance.

3.1.2 Interactive Data Cleaning

A number of commercial and research systems provide graphical interfaces leveraging the above methods. Many of these tools provide interfaces for schema matching, data integration, or entity resolution [14, 36, 54, 58, 69, 88, 103]. Toped^{++} [92] is an interface for creating *Topes*, objects that validate and transform data. Bellman [20] helps users understand the structure and quality of a database, but does not enable transformations. Topes support transformations such as text formatting and lookups, but provide little support for the filtering, reshaping, or aggregation operations of Wrangler. This system is also limited to finding formatting discrepancies for individual values. Profiler's data types are similar to *domains* in Potter's Wheel [86] and Scaffidi et al.'s *Topes* [92]. However, Profiler detects a broader range of discrepancies, including distribution-dependent outliers and duplicate values. Unlike these prior tools, Profiler also generates scalable interactive visual summaries to aid anomaly assessment.

Most closely related to our systems is prior work on interactive data cleaning. Potter's Wheel [86] provides a transformation language for data formatting and outlier detection. Wrangler extends the Potter's Wheel language with key differences discussed later. Ajax [28] also provides an interface to specify transforms, with advanced facilities for entity resolution. Neither tool provides much support for direct manipulation: interaction is largely restricted to menu-based commands or entering programming statements. Google Refine [48] (formerly Freebase GridWorks) leverages Freebase to enable entity resolution and

discrepancy detection. It provides summarization and filtering support through faceted histograms. Though users can specify some commands graphically, others must be written in a command language. Moreover, the system assumes that input data arrives in a proper tabular format, limiting the forms of data to which it can be applied. While Google Refine [49] supports both faceted browsing and text clustering to identify data quality issues, Refine users must manually specify which facets and clusters to create. In contrast, Profiler automatically suggests visualizations to aid discovery and assessment of discrepancies.

3.1.3 Editing and Auditing Transformations

Existing research in visualization highlights the value of explicitly recording the provenance of an analysis. For example, the VisTrails [8] system provides a general infrastructure for authoring and reviewing visualization workflows. VisTrails maintains a detailed history for each workflow and across versions, including the insertion, deletion, and parameterization of visualization operators. However, VisTrails, along with most other visualization history tools [39, 55, 65, 96], focuses on analysis and does not cover the process of data transformation necessary to use the visualization tools in the first place. More general scientific workflow tools [17, 72, 77] enable the creation and maintenance of workflows, but often by providing access to heterogeneous tools and scripting languages. Provenance-aware database systems [5] can track the lineage of data over multiple transformations and joins, but rarely support the steps necessary for transforming raw data into an appropriate format for import.

Informed by this prior work, we contend that the proper output of data wrangling is not just transformed data, but an *editable and auditable description* of the data transformations applied. High-level transformation descriptions will enable repeatability, modification, and recording of data provenance. Transforms could then be indexed and shared, enabling analysts to benefit from the work of others. Such transforms might also provide an artifact that can be annotated, enabling analysts to share their rationale for various data cleaning decisions.

3.2 Data Anomalies

Our work in Profiler (Chapter 6) draws on prior research to identify and classify common data quality issues and methods for automatically detecting these issues.

3.2.1 Anomaly Classification

The database and statistics literature includes many taxonomies of anomalous data [19, 23, 41, 62, 85, 102]. In Profiler (Chapter 6), we focus on errors that arise within a single relational table. Guided by prior taxonomies, we identified five categories of anomalies to address in Profiler:

Missing data results from a number of sources, including incomplete collection or redaction due to privacy concerns. Missing data can take the form of missing records or missing attributes. These issues can lead to a loss of statistical power if too many cases are unobserved and can introduce bias into model estimates, especially when data is not missing at random [2].

Erroneous data can arise because of error during data entry, measurement, or distillation [41]. Obviously, analysis of incorrect data can lead to incorrect conclusions.

Inconsistent data refers to variable encodings of the same value. Examples include variations in spelling or formatting, measurement units, or coding schemes (e.g., names vs. abbreviations).

Extreme values such as outliers can undermine robust analysis and may be erroneous. Extreme values may be standard univariate outliers, or may be type specific. For example, time-series outliers generally take two forms [102]: an additive outlier is an unexpected, transient movement in a measured value over time, whereas an innovation outlier is an unexpected movement that persists over time.

Key violations refer to data that violate primary key constraints. For example, having two employees with the same social security number violates the assumption that SSN is a key.

Observed issues can fall into multiple categories: a numeric outlier may result from an accurate measurement of an extreme value, a data entry error, or from inconsistent units (feet vs. meters).

3.2.2 Automated Anomaly Detection

The database and machine learning communities have also contributed a number of algorithmic techniques for aiding data cleaning and integration. These techniques include methods for detecting outliers [11, 41, 45, 102], detecting erroneous values [41, 45], information extraction [4, 98], entity resolution [23], type inference [26], key violations [47], and schema matching [36, 84]. While these routines flag potential issues, most types of error require some form of human intervention to assess and correct [62]. In Wrangler and Profiler we seek to surface such techniques in an accessible manner. For instance, in Profiler we generate scalable interactive visual summaries to aid anomaly assessment.

3.3 Visual Analysis

Profiler (Chapter 6) leverages and extends visualization techniques to support assessment of the automated routines for anomaly detection discussed above (Section 3.2). The system also builds upon techniques used in existing visual analytic systems.

3.3.1 Visualization at Scale

Visualization can support discovery of patterns in data, including anomalies [61]. Aggregation, clustering and sorting have been used in various contexts to support scalable visualization for large data sets [10, 64, 89, 105]. A common recourse is to apply aggregation, but doing so risks obscuring low-level details in the data. Histograms are a common form of 1D aggregation, both for categorical data and for binned quantitative data. Binning is also applicable in scatter plots, for example to form a heat map visualizing data density. Statisticians have suggested numerous techniques for plotting data at scale [105, 9], including using hexagonal (as opposed to rectangular) 2D bins in order to improve density estimates and deemphasize horizontal and vertical striping [9]. Techniques such as online aggregation [42] might also be applied: a visualization may show a dynamic aggregate of a sample, with error bars indicating a confidence interval. As query processing continues, the visualization can update the computed values and intervals; the analyst need not wait until completion to assess the data and proceed on to other tasks. While initially proposed for

1D quantitative data, such dynamic sampling-based techniques might be extended to other data types. More research is necessary to characterize the strengths and limits of such approaches. Through linked highlighting ("brushing & linking"), coordinated multiple views enable assessment of relationships between data dimensions [76, 108]. Profiler's visualization layer extends this prior work with a set of type-specific aggregate visualizations that aid assessment of data quality issues.

3.3.2 Visualizing Missing Data

Some prior work has investigated what forms of visual encoding or annotation should be used to flag known data quality issues during visual analysis. Eaton et al. [21] categorize visualization techniques based on how amenable they are to surfacing missing data. In a time-series chart, the use of spatial position along the time axis allows visualization of missing data with a gap or icon. In other spatial domains, such as maps or fluid flows, color interpolation techniques might be applied. For example, Restorer [104] maintains smooth luminance contours but drops hue to unobtrusively show missing values. However, space-filling visualizations such as pie charts or treemaps will obscure the presence of missing data and may bias the appearance of other items.

Eaton et al. [21] report on a user study that compared three design variants to represent missing data. They find that users do not necessarily realize that data is missing when it is replaced by a default value. The rate of error might be reduced by cues that more explicitly highlight imputed elements. They also find that even if the missing data is noticeable, users regularly make general conclusions with the remaining partial data. This study provides evidence for a need to indicate the presence of missing information. Both Wrangler and Profiler use data quality meters to indicate the proportion of missing or erroneous data in a column.

3.3.3 Visualizing Uncertain Data

Uncertainty arises from a number of sources, including measurement error (e.g., sensor drift), missing data, and aggregations or sampling of data sets. Skeels et al. [97] create a classification of uncertainty informed by prior work and an interview-based user study. In

their classification, they identify five types of uncertainty—Measurement Precision, Completeness, Inference, Disagreement, and Credibility. A number of techniques have been developed to visualize uncertain data [16, 32, 79, 81]. They often employ a variety of visual encodings, including transparency, blur, error bars, and error ellipses. Olston and Mackinlay [79] describe mechanisms for visualizing uncertain data with known bounds. CandidTree shows two types of structural uncertainty using color and transparency based on the differences between two tree structures [68]. Other techniques include adding glyphs [16], adding or modifying geometry [33], animation [30], and sonification [70]. For example, LISTEN visualizes the geometric uncertainty using sound, which represents the difference between geometric quantities obtained by two interpolants [70].

3.3.4 Visual Analytic Tools

Visual analytic tools such as Tableau [100], GGobi [101], and Improvise [108] enable analysts to construct multi-dimensional views of data. However, these tools generally require users to choose which variables to visualize. As the number of data subsets explodes combinatorially, analysts must often rely on significant domain expertise to identify variables that may contain or help explain anomalies. To facilitate the view selection process, Profiler automatically suggests both data subsets and appropriate summary visualizations based on identified anomalies and inferred data types. While other tools support general exploratory analysis, Profiler provides guided analytics to enable rapid quality assessment.

Others have explored interfaces for guiding analysis and suggesting appropriate views. Social Action [82] uses a wizard-like interface to guide users through social network analysis. Seo and Shneiderman's rank-by-feature framework [94] sorts histograms and scatterplots of numeric data according to user-selected criteria. Others have used dimensionality reduction, clustering and sorting to aid visualization of multidimensional data [34, 44, 110]. In Profiler, we use anomaly detection followed by mutual information analysis to suggest a set of coordinated summary views for assessing data quality issues. Our suggestion engine automates the choice of data columns, aggregation functions and visual encodings.

Chapter 4

Wrangler: Visual Data Transformation

As discussed in Chapter 2, analysts expend an inordinate amount of time and effort manipulating data and assessing data quality issues. Such "data wrangling" regularly involves reformatting data values or layout, correcting erroneous or missing values, and integrating multiple data sources. These transforms are often difficult to specify and difficult to reuse across analysis tasks, teams, and tools. In response, we introduce *Wrangler*, an interactive system for creating data transformations. The primary contribution of Wrangler is to combine combine direct manipulation of visualized data with automatic inference of relevant transforms, enabling analysts to iteratively explore the space of applicable operations and preview their effects. Wrangler leverages semantic data types (e.g., geographic locations, dates, classification codes) to aid validation and type conversion. Interactive histories support review, refinement, and annotation of transformation scripts. User study results show that Wrangler significantly reduces specification time and promotes the use of robust, auditable transforms instead of manual editing.

4.1 Introduction

Despite significant advances in technologies for data management and analysis, it remains time-consuming to inspect a data set and mold it to a form that allows meaningful analysis to begin. Analysts must regularly restructure data to make it palatable to databases, statistics packages, and visualization tools. To improve data quality, analysts must also

identify and address issues such as misspellings, missing data, unresolved duplicates, and outliers. Our own informal interviews with data analysts have found that these types of transforms constitute the most tedious component of their analytic process. Others estimate that data cleaning is responsible for up to 80% of the development time and cost in data warehousing projects [19]. Such "data wrangling" often requires writing idiosyncratic scripts in programming languages such as Python and Perl, or extensive manual editing using interactive tools such as Microsoft Excel. Moreover, this hurdle discourages many people from working with data in the first place. Sadly, when it comes to the practice of data analysis, "the tedium is the message."

Part of the problem is that reformatting and validating data requires transforms that can be difficult to specify and evaluate. For instance, analysts often split data into meaningful records and attributes—or validate fields such as dates and addresses—using complex regular expressions that are error-prone and tedious to interpret [6, 92]. Converting coded values, such as mapping FIPS codes to U.S. state names, requires integrating data from one or more external tables. The effects of transforms that aggregate data or rearrange data layout can be particularly hard to conceptualize ahead of time. As data sets grow in size and complexity, discovering data quality issues may be as difficult as correcting them.

Of course, transforming and cleaning a data set is only one step in the larger data lifecycle. Data updates and evolving schemas often necessitate the reuse and revision of transformations. Multiple analysts might use transformed data and wish to review or refine the transformations that were previously applied; the importance of capturing data provenance is magnified when data and scripts are shared. As a result, we contend that the proper output of data wrangling is not just transformed data, but an editable and auditable description of the data transformations applied.

This chapter presents the design of *Wrangler*, a system for interactive data transformation. We designed Wrangler to help analysts author expressive transformations while simplifying specification and minimizing manual repetition. To do so, Wrangler couples a *mixed-initiative user interface* with an underlying *declarative transformation language*.

With Wrangler, analysts specify transformations by building up a sequence of basic transforms. As users select data, Wrangler *suggests applicable transforms* based on the current context of interaction. Programming-by-demonstration techniques help analysts

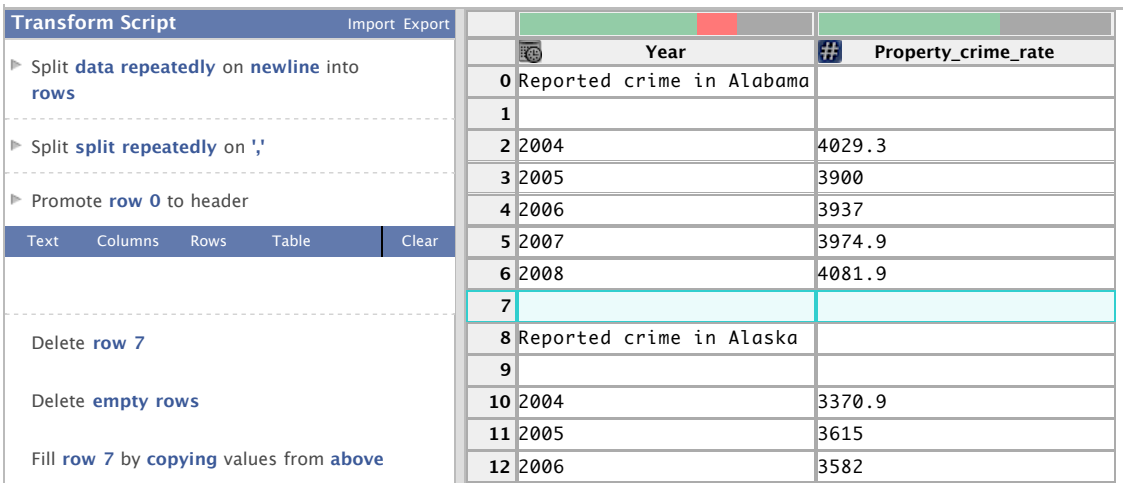


Figure 4.1: The Wrangler Interface. The left panel contains (from top-to-bottom) a history of transforms, a transform selection menu, and automatically suggested transforms based on the current selection. Bold text within the transform descriptions indicate parameters that can be clicked and revised. The right panel contains an interactive data table; above each column is a data quality meter.

specify complex criteria such as regular expressions. To ensure relevance, Wrangler enumerates and rank-orders possible transforms using a model that incorporates user input with the frequency, diversity, and specification difficulty of applicable transform types. To convey the effects of data transforms, Wrangler provides short *natural language descriptions*—which users can refine via interactive parameters—and *visual previews* of transform results. These techniques enable analysts to rapidly navigate and assess the space of viable transforms.

As analysts transform data, their steps are recorded in a script to facilitate reuse and provide documentation of data provenance. Wrangler's *interactive history viewer* supports review, refinement, and annotation of these scripts. Wrangler's high-level language supports a variety of runtime platforms: Wrangler scripts can be run in a web browser using JavaScript or translated into MapReduce or Python code.

We also conducted a controlled user study comparing Wrangler and Excel across a set of data wrangling tasks. We find that Wrangler significantly reduces specification time and promotes the use of robust transforms rather than manual editing.

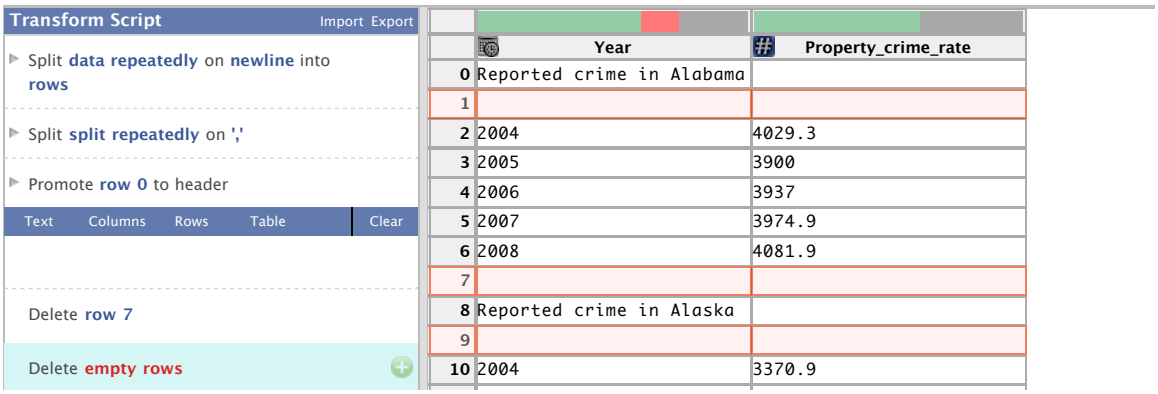


Figure 4.2: Row deletion. The analyst selects an empty row and chooses a *delete* transform. Red highlights preview which rows will be deleted.

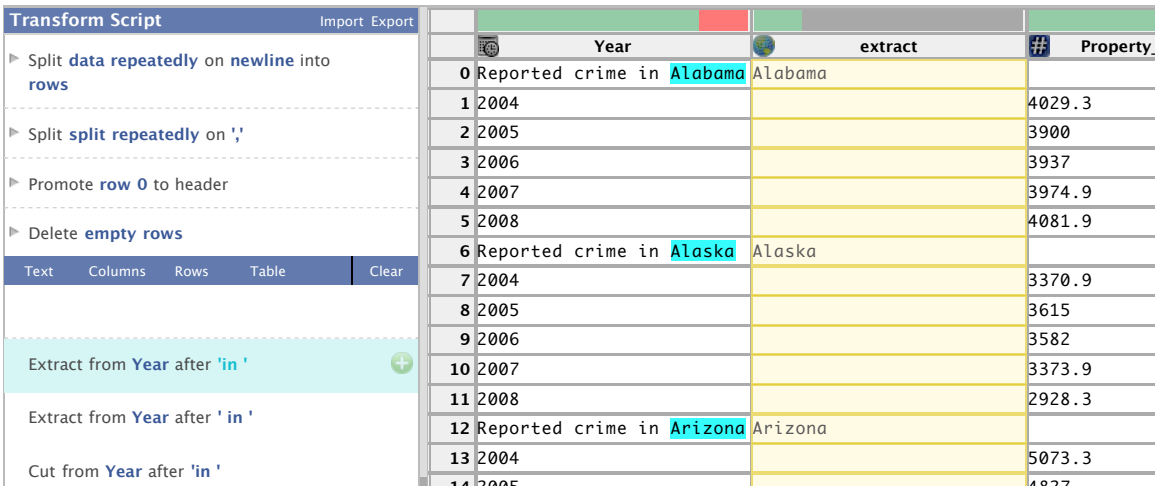


Figure 4.3: Text extraction. The analyst selects state names to *extract* them into a new column. Yellow highlights show a preview of the result.

4.2 Usage Scenario

Consider an example wrangling task, using housing crime data from the U.S. Bureau of Justice Statistics. The data were downloaded as a CSV (comma-separated values) file, but are not immediately usable by other tools: the data contains empty lines, U.S. states are organized in disjoint matrices, and the state names are embedded in other text. We describe how an analyst can use Wrangler to transform the data into more usable formats (Figures 4.1–4.7).

The analyst begins by pasting the text of the file into an input box; alternatively, she could upload the file. The interface now shows a data table (Fig. 4.1). To the left of the table is a panel containing an interactive history, a transform menu, and a transform editor. The history already contains three transforms, as Wrangler inferred that the data was in CSV format and so split the text into rows on newline characters, split the rows into columns on commas, and promoted the first row to be the table header. Note that the analyst could undo any transform by clicking the red undo button (which appears upon mouse-over of a transform), or could modify transform parameters in place. In this case, she has no need.

The analyst then begins wrangling the file into a usable form. The analyst could specify transforms explicitly by selecting a transform type from the menu and then assigning values to parameters; however, she instead opts to use direct manipulation along with Wrangler's suggestion engine. First, she *clicks* a row header for an empty row (7) to select it; the transformation editor suggests possible operations in response (Fig. 4.1). The first suggestion is to delete just the selected row. The analyst can navigate the suggestions using the keyboard *up* and *down* arrows or by mousing over the description in the editor pane. As she navigates the suggestions, Wrangler previews the effects of the transforms in the data table. For deletions, the preview highlights the candidate deleted rows in red (Fig. 4.2). The analyst mouses over the suggestion to delete all empty rows in the table and clicks the green *add* button to execute the transform. The system then adds the deletion operation to the history view.

The analyst would like to compare data across states, so she now needs to extract the state names and add them to each row of the data. She selects the text 'Alaska' in row 6 of the "Year" column. Wrangler initially interprets this as selecting text at positions 18-24. The analyst updates Wrangler's inference by selecting 'Arizona' in the cell six rows below. Wrangler now suggests extracting text occurring after the string "in " (Fig. 4.3). The analyst executes this transform and renames the resulting column "State". She notices that the column is sparsely populated. These missing values are indicated by the gray bar in the *data quality meter* above the column. The analyst clicks the gray bar and Wrangler suggests transforms for missing values. The analyst chooses to fill empty cells with the value from above (Fig. 4.4).

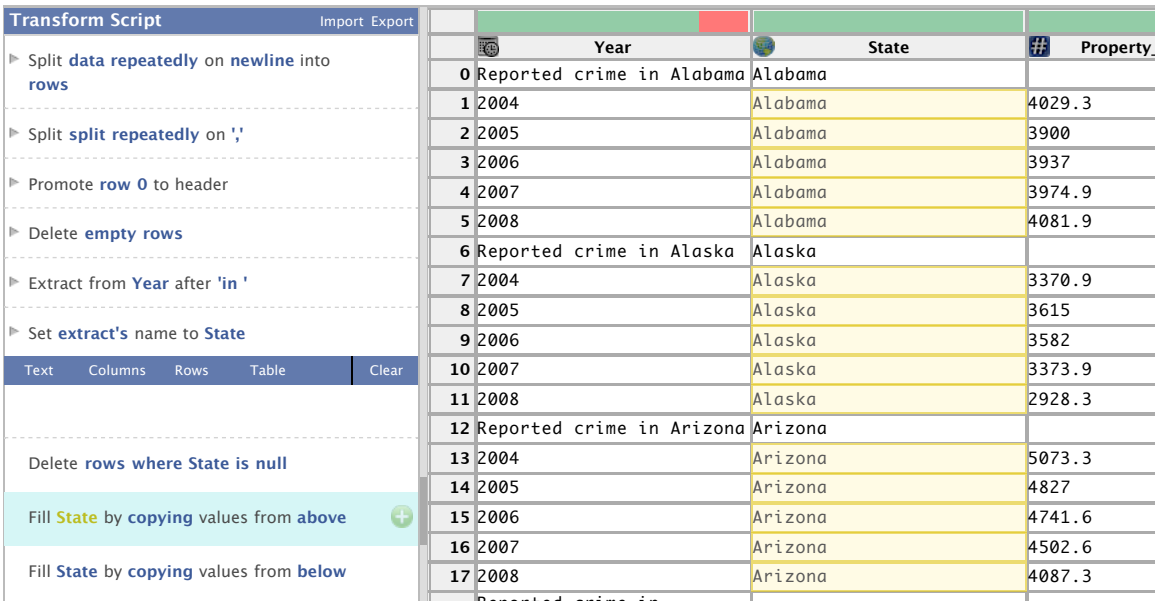


Figure 4.4: Filling missing values. The analyst populates empty cells by clicking the gray bar (Fig. 3) in the data quality meter above the "State" column, and then selecting a *fill* transform.

Looking at the "Year" column, the analyst notices a red bar in the data quality meter indicating inconsistent data types. Wrangler has inferred that the column primarily contains numbers, and so has flagged non-numeric values as potential errors. She decides to remove the rows containing the text 'Reported'. She selects the text 'Reported' in row 0. Wrangler suggests *split*, *extract*, and *cut* transforms, but no delete operations. In response, the analyst selects the *Delete* command from the *Rows* menu in the transform editor. This action reorders the suggestions so that *delete* commands have higher ranking. She finds the suggestion that deletes the unwanted rows (Fig. 4.5) and executes the transform.

At this point the analyst has wrangled the data into a proper relational format, sufficient for export to database and visualization tools. But now suppose she would like to create a cross-tabulation of crime rates by state and year for subsequent graphing in Excel. She selects the "Year" and "Property_crime_rate" columns, previews the suggested *unfold* operation (Fig. 4.6), then executes it to create the desired cross-tab. The *unfold* operation creates new columns for each unique value found in the "Year" column, and reorganizes the "Property_crime_rate" values by placing each in the appropriate cell in the resulting

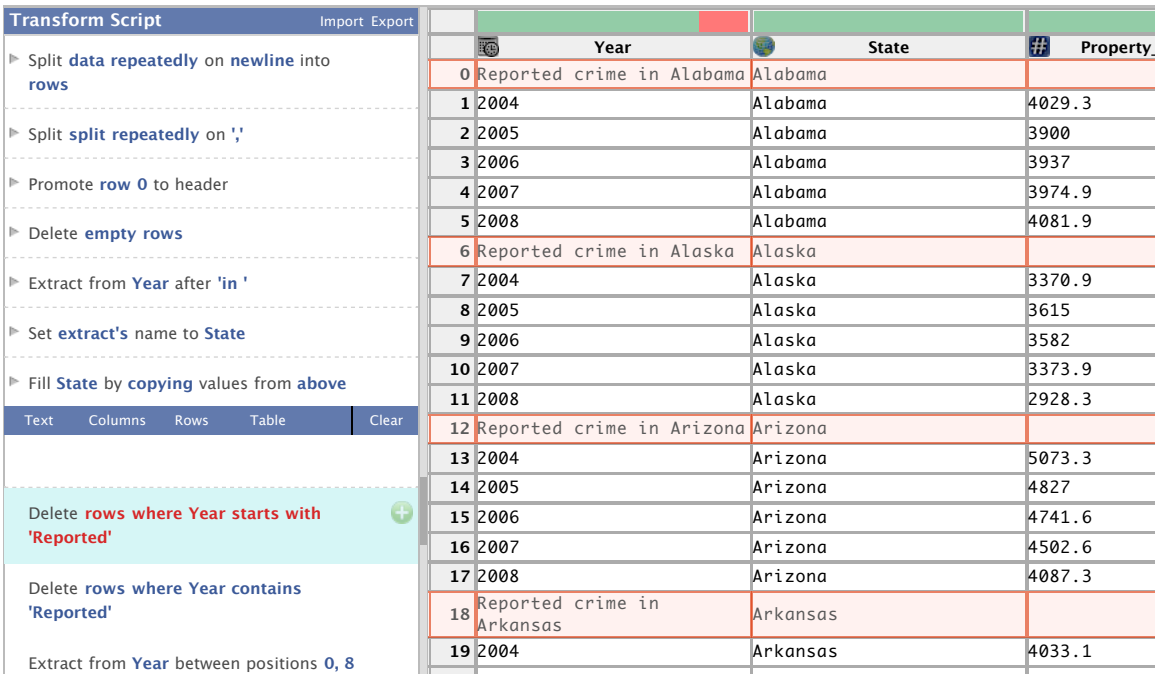


Figure 4.5: Deleting rows. The analyst selects text in an unwanted row and selects a *delete* operation within the "Rows" menu. Red highlighting previews which rows will be deleted.

matrix.

The analyst's process results in a transformation script written in a declarative transformation language. The script provides an auditable description of the transformation enabling later inspection, reuse, and modification. The analyst can also annotate these transformations with her rationale. By clicking the *Export* button above the transformation history, the analyst can either save the transformed data or generate runnable code implementing the transformation (Fig. 4.7).

4.3 Design Process

We based Wrangler on a transformation language with a handful of operators. Originally we thought that each of these operators might correspond to a single interaction with example data in a table view. However, after considering different mappings and evaluating

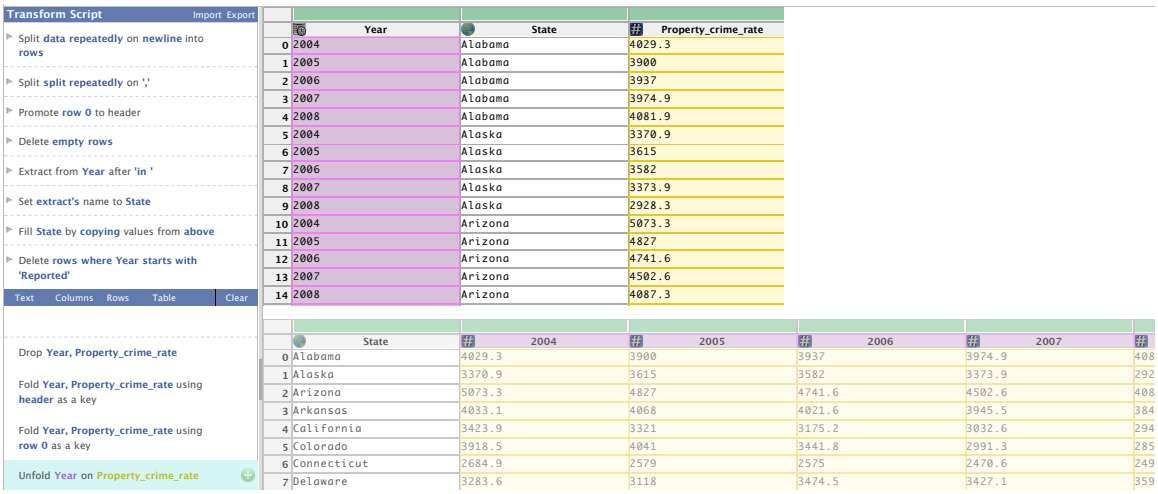


Figure 4.6: Table reshaping. The analyst selects two columns, and then elects to *unfold* them to create a cross-tabulation. A ghosted table overlay previews the result. Color highlights show the correspondence of data between the start and end states.

their implications, we were unable to devise an intuitive and unambiguous mapping between simple gestures and the full expressiveness of the language. A given interaction could imply multiple transforms and multiple interactions might imply the same transform.

Although this many-to-many relationship between the language and interaction might complicate our interface, we found the relationship to be relatively sparse in practice: the number of likely transforms for a given gesture is small. As a result, we adopted a *mixed-initiative* approach [46]; instead of mapping an interaction to a single transform, we surface likely transforms as an ordered list of suggestions. We then focused on rapid means for users to navigate—prune, refine, and evaluate—these suggestions to find a desired transform.

Wrangler is a browser-based web application, written in Java-Script. In the next section we describe the Wrangler transformation language. We then present the Wrangler interface and its techniques for navigating suggestion space. Next, we describe Wrangler's mechanisms for verification. We go on to discuss the technical details of our inference engine.

```
split('data').on(NEWLINE).max_splits(NO_MAX)
split('split').on(COMMA).max_splits(NO_MAX)
columnName().row(0)
delete(isEmpty())
extract('Year').on(/.*/).after(/in /)
columnName('extract').to('State')
fill('State').method(COPY).direction(DOWN)
delete('Year starts with "Reported"')
unfold('Year').above('Property_crime_rate')
```

Figure 4.7: The result of a data wrangling session is a declarative data cleaning script, shown here as generated JavaScript code. The script encodes a step-by-step description of how to operate on input data; a Wrangler runtime evaluates the script to produce transformed data.

4.4 The Wrangler Transformation Language

Underlying the Wrangler interface is a declarative data transformation language 4.1. Both prior work [28, 67, 86] and empirical data guided the language design. As our starting point we used the Potter's Wheel transformation language [86] (which in turn draws from SchemaSQL [67]). Informed by a corpus of data sets gathered from varied sources (e.g., data.gov, NGOs, log files, web APIs), we then extended the language with additional operators for common data cleaning tasks. These include features such as positional operators, aggregation, semantic roles, and complex reshaping operators (e.g., using multiple key rows for cross-tabs). We also introduced conditional mapping operators (e.g., update country to "U.S." where state="California"). Language statements manipulate *data tables* with numbered rows and named columns of data. Wrangler treats raw text as a "degenerate" table containing one row and one column. The language consists of eight classes of transforms, described below.

Map transforms map one input data row to zero, one, or multiple output rows. *Delete* transforms (one-to-zero) accept predicates determining which rows to remove. One-to-one transforms include *extracting*, *cutting*, and *splitting* values into multiple columns; reformatting; simple arithmetic; and value *updates*. One-to-many transforms include operations for splitting data into multiple rows, such as splitting a text file on newlines or unnesting

Transform	Description
Cut	Remove selected text from cells in specified columns.
Delete	Remove rows that match given indices or predicates.
Drop	Remove specified columns from the table.
Edit	Edit the text in each cell of the specified columns.
Extract	Copy text from cells in a column into a new column.
Fill	Fill empty cells using values from adjacent cells.
Fold	Reshape a table into columns of key-value sets; selected rows map to keys, selected columns to values.
Merge	Concatenate multiple columns into a single column.
Promote	Promote row values to be the column names.
Split	Split a column into multiple columns by delimiters.
Translate	Shift the position of cell values by a given offset.
Transpose	Transpose the rows and columns of the table.
Unfold	Reshape a table by mapping key-value sets to a collection of new columns, one per unique key.

Table 4.1: The Wrangler Transformation Language. Each transform accepts as parameters some combination of enumerable values and text, row, or column selection criteria.

arrays and sets.

Lookups and joins incorporate data from external tables. Wrangler includes extensible lookup tables to support common types of transformations, such as mapping zip codes to state names for aggregation across states. Currently Wrangler supports two types of joins: equi-joins and approximate joins using string edit distance. These joins are useful for lookups and for correcting typos for known data types.

Reshape transforms manipulate table structure and schema. Wrangler provides two reshaping operators: *fold* and *unfold*. *Fold* collapses multiple columns to two or more columns containing key-value sets, while an *unfold* creates new column headers from data values; see [86] for an extended discussion. Reshaping enables higher-order data restructuring and is common in tools such as R and Excel Pivot Tables.

Positional transforms include *fill* and *lag* operations. *Fill* operations generate values based on neighboring values in a row or column and so depend on the sort order of the table. For example, an analyst might fill empty cells with preceding non-empty values. The *lag* operator shifts the values of a column up or down by a specified number of rows.

The language also includes functions for **sorting**, **aggregation** (e.g., sum, min, max, mean, standard deviation), and **key generation** (a.k.a., *skolemization*). Finally, the language contains **schema** transforms to set column names, specify column data types, and assign semantic roles.

To aid data validation and transformation, Wrangler supports standard *data types* (e.g., integers, numbers, strings) and higher-level *semantic roles* (e.g., geographic location, classification codes, currencies). Data types comprise standard primitives and associated parsing functions. Semantic roles consist of additional functions for parsing and formatting values, plus zero or more transformation functions that map between related roles. As an example, consider a semantic role defining a *zip code*. The zip code role can check that a zip code parses correctly (i.e., is a 5 digit number) and that it is a valid zip code (checking against an external dictionary of known zipcodes). The zip code role can also register mapping functions, e.g., to return the containing state or a central lat-lon coordinate. Wrangler leverages types and roles for parsing, validation, and transform suggestion. The Wrangler semantic role system is extensible, but currently supports a limited set of common roles such as geographic locations, government codes, currencies, and dates.

The Wrangler language design co-evolved with the interface described in subsequent sections. We sought a consistent mapping between the transforms shown in the interface and statements in the language. Disconnects between the two might cause confusion [75], particularly when analysts try to interpret code-generated scripts. As a result, we chose to introduce redundancy in the language by adding operators for high-level actions that are commonly needed but have unintuitive lower-level realizations (e.g., *positional* operators can be realized using *key* transforms, self-joins, and scalar functions). The result is a clear one-to-one mapping between transforms presented in the interface and statements in output scripts. Prior work [67, 86] proves that our basic set of transforms is sufficient to handle all one-to-one and one-to-many transforms. Through both our own practice and discussions with analysts, we believe our extended language is sufficient to handle a large variety of data wrangling tasks.

4.5 The Wrangler Interface Design

The goal of the Wrangler interface is to enable analysts to author expressive transformations with minimal difficulty and tedium. To this aim, our interface combines direct manipulation, automatic suggestion, menu-based transform selection, and manual editing of transform parameters. This synthesis of techniques enables analysts to navigate the space of transforms using the means they find most convenient.

Both novices and experts can find it difficult to specify transform parameters such as regular expressions. While direct manipulation selections can help, inference is required to suggest transforms without programming. To reduce this gulf of execution [75], Wrangler uses an *inference engine* that suggests data transformations based on user input, data type or semantic role, and a number of empirically-derived heuristics. These suggestions are intended to facilitate the discovery and application of more complicated transforms.

However, suggested transforms (and their consequences) may be difficult to understand. To reduce this gulf of evaluation [75], Wrangler provides *natural language descriptions* and *visual transform previews*. Natural language descriptions are intended to enhance analysts' ability to review and refine transformation steps. Textual annotations enable communication of analyst intent. Wrangler also couples verification (run in the background as data is transformed) with visualization to help users discover data quality issues.

4.5.1 Basic Interactions

The Wrangler interface supports six basic interactions within the data table. Users can select rows, select columns, click bars in the data quality meter, select text within a cell, edit data values within the table (for mass editing [50, 73]), and assign column names, data types or semantic roles. Users can also choose transforms from the menu or refine suggestions by editing transform descriptions as described below.

4.5.2 Automated Transformation Suggestions

As a user interacts with data, Wrangler generates a list of *suggested transforms*. In some cases the set of possible suggestions is large (in the hundreds), but we wish to show only

a relevant handful to avoid overload. Instead of enumerating the entire suggestion space, users can prune and reorder the space in three ways. First, users can provide more examples to disambiguate input to the inference engine. Providing examples is especially effective for text selections needed for splitting, extraction, and reformatting; two or three well-chosen examples typically suffice. Second, users can filter the space of transforms by selecting an operator from the transform menu. Third, users can edit a transform by altering the parameters of a transform to a desired state.

Wrangler does not immediately execute a selected suggestion. Instead, Wrangler makes it the *current working transform*. The user can edit this transform directly; as a user edits parameters, the suggestion space updates to reflect these edits. Also, a user can instead interact with the table to generate new suggestions that use the working transform as context.

4.5.3 Natural Language Descriptions

To aid apprehension of suggested transforms, Wrangler generates short *natural language descriptions* of the transform type and parameters. These descriptions are editable, with parameters presented as bold hyperlinks (Fig. 4.8). Clicking a link reveals an in-place editor appropriate to the parameter (Fig. 4.8b). Enumerable variables (such as the direction of a fill) are mapped to drop-down menus while free-form text parameters are mapped to text editors with autocomplete.

We designed these descriptions to be concise; default parameters that are not critical to understanding may be omitted. For example, the *unless between* parameter for split operations indicates regions of text to ignore while splitting. In most cases, this parameter is left undefined and including it would bloat the description. To edit hidden parameters, users can click the expansion arrow to the left of the description, revealing an editor with entries for all possible parameters.

We also sought to make parameters within descriptions readable by non-experts. For instance, we translate regular expressions into natural language via pattern substitution (e.g., (\d+) to ‘number’). This translation can make some descriptions less concise but increases readability. Translation is only performed with regular expressions generated by the Wrangler inference engine. If a user types in a custom expression, Wrangler will reflect

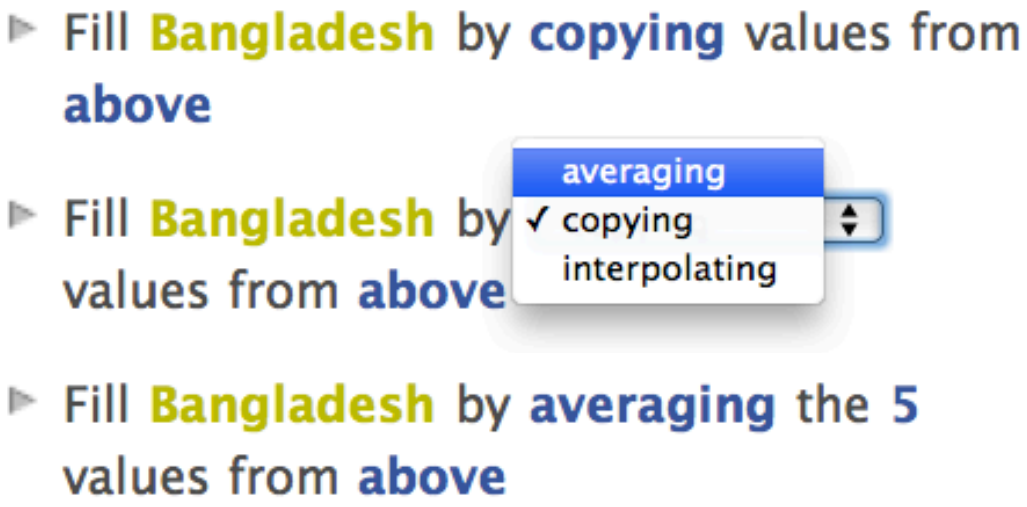


Figure 4.8: Editable Natural Language Descriptions. (a) An example of an editable description; highlighted text indicates editable parameters. (b) Clicking on a parameter reveals an in-place editor. (c) After editing, the description may update to include new parameters. In this case, a new window size parameter is displayed for the moving average.

their input.

4.5.4 Visual Transformation Previews

Wrangler uses *visual previews* to enable users to quickly evaluate the effect of a transform. For most transforms, Wrangler displays these previews in the source data, and not as a separate visualization (e.g., side-by-side before and after views). In-place previews provide a visual economy that serves a number of goals. First, displaying two versions of a table inherently forces both versions to be small, which is particularly frustrating when the differences are sparse. Second, presenting in-place modifications draws user attention to the effect of the transformation in its original context, without requiring a shift in focus across multiple tables. As we discuss next, in-place previews better afford direct manipulation for users to revise the current transform.

Wrangler maps transforms to at least one of five preview classes: selection, deletion, update, column and table. In defining these mappings, we attempted to convey a transform's effect with minimum displacement of the original data. This stability allows users

to continue interacting with the original data, e.g., to provide new selection examples.

Selection previews highlight relevant regions of text in all affected cells (Fig. 4.3). Deletion previews color to-be-deleted cells in red (Fig. 4.2). Update previews overwrite values in a column and indicate differences with yellow highlights (Fig. 4.4). Column previews display new derived columns, e.g., as results from an *extract* operation (Fig. 4.3). We show a side-by-side display of versions when previewing *fold* and *unfold* transforms. These alter the structure of the table to such an extent that the best preview is to show another table (Fig. 4.6, 4.9). These table previews use color highlights to match input data to their new locations in the output table. Some transforms map to multiple classes; e.g., *extract* transforms use both selection and column previews.

When possible, previews also indicate where the user can modify the transform through either direct manipulation or description refinement. Highlighting selected text or cells works well for certain transformations. For example, by highlighting the text selected by a regular expression in each cell, users can determine which examples they need to fix. For reshape transforms, Wrangler highlights the input data in the same color as the corresponding output in the secondary table. For instance, in a *fold* operation, data values that will become keys are colored to match the keys in the output table (Fig. 4.9). Wrangler also highlights the parameters in the transform description using the same colors as those generated in previews (Fig. 4.3–4.6). The consistent use of colors allows users to associate clauses in a description with their effects in the table.

4.5.5 Transformation Histories and Export

As successive transforms are applied, Wrangler adds their descriptions to an interactive *transformation history viewer*. Users can edit individual transform descriptions and selectively enable and disable prior transforms. Upon changes, Wrangler runs the edited script and updates the data table. Toggling or editing a transform may result in downstream errors; Wrangler highlights broken transforms in red and provides an error message to aid debugging.

Wrangler scripts also support lightweight text annotations. Analysts can use annotations to document their rationale for a particular transform and may help future users better

split	split1	split2	split3	split4
	2004	2004	2004	2003
STATE	Participation Rate 2004	Mean SAT I Verbal	Mean SAT I Math	Participation Rate
New York	87	497	510	82
Connecticut	85	515	515	84
Massachusetts	85	518	523	82
New Jersey	83	501	514	85
New Hampshire	80	522	521	75
D.C.	77	489	476	77
Maine	76	505	501	70
Pennsylvania	74	501	502	73
Delaware	73	500	499	73
Georgia	73	494	493	66

split	fold	fold1	value
New York	2004	Participation Rate 2004	87
New York	2004	Mean SAT I Verbal	497
New York	2004	Mean SAT I Math	510
New York	2003	Participation Rate 2003	82
New York	2003	Mean SAT I Verbal	496
New York	2003	Mean SAT I Math	510
Connecticut	2004	Participation Rate 2004	85
Connecticut	2004	Mean SAT I Verbal	515
Connecticut	2004	Mean SAT I Math	515
Connecticut	2003	Participation Rate 2003	84
Connecticut	2003	Mean SAT I Verbal	512

Figure 4.9: Visual preview of a *fold* operation. For transforms that rearrange table layout, Wrangler previews the output table and uses color highlights to show the correspondence of values across table states.

understand data provenance. To annotate a transform, users can click the *edit* icon next to the desired transform and write their annotation in the resulting text editor. Users can view an annotation by mousing over the same *edit* icon. These annotations appear as comments in code-generated scripts. Users can export both generated scripts and transformed data; clicking the *Export* button in the transform history invokes export options. Analysts can later run saved or exported scripts on new data sources, modifying the script as needed.

4.6 Types, Roles, and Verification

It is often difficult to discover data quality issues and therefore difficult to address them by constructing the appropriate transform. Wrangler aids discovery of data quality issues through the use of data types and semantic roles.

As users transform data, Wrangler attempts to infer the data type and semantic role for

each column. Wrangler applies validation functions to a sample of a column's data to infer these types, assigning the type that validates for over half of the non-missing values. When multiple types satisfy this criteria, Wrangler assigns the more specific one (e.g., *integer* is more specific than *double*). Wrangler infers semantic roles analogously. An icon in the column header indicates the semantic role of the column, or the underlying data type if no role has been assigned. Clicking the icon reveals a menu with which users can manually assign a type or role.

Above each column is a *data quality meter*: a divided bar chart that indicates the proportion of values in the column that verify completely. Values that parse successfully are indicated in green; values that match the type but do not match the role (e.g., a 6 digit zip code) are shown in yellow; those that do not match the type (e.g., 'One' does not parse as an integer) are shown in red; and missing data are shown in gray. Clicking a bar generates suggested transforms for that category. For instance, clicking the *missing values* bar will suggest transforms to fill in missing values or delete those rows. Clicking the *fails role* bar will suggest transforms such as a similarity join on misspelled country names.

4.7 The Wrangler Inference Engine

We now present the design of the Wrangler inference engine, which is responsible for generating a ranked list of suggested transforms. Inputs to the engine consist of user interactions; the current working transform; data descriptions such as column data types, semantic roles, and summary statistics; and a corpus of historical usage statistics. Transform suggestion proceeds in three phases: inferring transform parameters from user interactions, generating candidate transforms from inferred parameters, and finally ranking the results.

4.7.1 Usage Corpus and Transform Equivalence

To generate and rank transforms, Wrangler's inference engine relies on a corpus of usage statistics. The corpus consists of frequency counts of transform descriptors and initiating interactions. We built our initial corpus by wrangling our collection of gathered data sets. The corpus updates over time as more analysts use Wrangler.

For any given transform, we are unlikely to find an exact match in the corpus. For instance, an analyst may perform a *fold* operation over a combination of columns and rows that does not appear in the corpus. In order to get useful transform frequencies, we define a relaxed matching routine: two transforms are considered *equivalent* in our corpus if (a) they have an identical transform type (e.g., *extract* or *fold*) and (b) they have equivalent parameters as defined below.

Wrangler transforms accept four basic types of parameters: row, column or text selections and enumerables. We treat two row selections as equivalent if they both (a) contain filtering conditions (either index- or predicate-based) or (b) match all rows in a table. Column selections are equivalent if they refer to columns with the same data type or semantic role. We based this rule on the observation that transforms that operate on identical data types are more likely to be similar. Text selections are equivalent if both (a) are index-based selections or (b) contain regular expressions. We consider enumerable parameters equivalent only if they match exactly. We chose these equivalency classes based on exploratory analysis of our corpus and they seem to work well in practice. As our corpus of transforms grows with more use, we plan to explore more principled approaches (such as clustering) to refine our matching routines.

4.7.2 Inferring Parameter Sets from User Interaction

In response to user interaction, Wrangler attempts to infer three types of transform parameters: row, column, or text selections. For each type we enumerate possible parameter values, resulting in a collection of inferred parameter sets. We infer a parameter's values independent of the other parameters. For example, we infer regular expressions for text selection based solely on the selected text, a process otherwise independent of which rows or columns are selected.

We infer row selections based on row indices and predicate matching. We list predicates of the form "row is empty" and "column [equals | starts with | ends with | contains] selected-value", then emit the selections that match the rows and text currently selected in the interface. For column selections we simply return the columns that users have interacted with.

Emitted text selections are either simple index ranges (based directly on selections in the interface) or inferred regular expressions. To generate regular expressions, we tokenize the text within a cell and extract both the selected text and any surrounding text within a 5 token window. We annotate tokens with one or more labels of the form *number*, *word*, *uppercase* word, *lowercase* word, or *whitespace*. We then enumerate label sequences that match the text before, within, and after the selection range (see Fig. 4.10); sequences can contain either an annotation label or the exact token text.

Next we emit all possible combinations of before, within, and after sequences that match all current text selection examples in the interface. It is then straightforward to translate matching label sequences into regular expressions.

4.7.3 Generating Suggested Transforms

After inferring parameter sets, Wrangler generates a list of transform suggestions. For each parameter set, we loop over each transform type in the language, emitting the types that can accept all parameters in the set.

For example, a *split* transform can accept a parameter set containing a text selection, but an *unfold* transform can not. Wrangler instantiates each emitted transform with parameters from the parameter set. To determine values for missing parameters, we query the corpus for the top-k (default 4) parameterizations that co-occur most frequently with the provided parameter set. During this process we do not infer complex criteria such as row predicates or regular expressions; we do infer enumerable parameters, index-based row selections, and column inputs. We then filter the suggestion set to remove "degenerate" (no-op) transforms that would have no effect on the data.

4.7.4 Ranking Suggested Transforms

Wrangler then rank-orders transform suggestions according to five criteria. The first three criteria rank transforms by their type; the remaining two rank transforms within types. Ensuring that transforms of the same type are adjacent helps users compare varying parameterizations more easily.

First, we consider explicit interactions: if a user chooses a transform from the menu

(a) `Reported crime in Alabama`

(b)

before:	{'in', ' '}	'Alabama' → {'Alabama', *word*}
selection:	{'Alabama'}	'in' → {'in', *word*, *lowercase*}
after:	∅	' ' → {' '}

(c)

before:	{(' '), ('in', ' '), (*word*, ' '), (*lowercase*, ' ')}
selection:	{('Alabama'), (*word*)}
after:	∅

(d)

{(),('Alabama'),()}	~~{(),(*word*),()}~~
~~{(' '),(),()}~~	~~{(*word*, ' '),(),()}~~
{(' '),('Alabama'),()}	{(*word*, ' '),('Alabama'),()}
~~{(' '),(*word*),()}~~	~~{(*word*, ' '),(*word*),()}~~
{('in', ' '),(),()}	~~{(*lowercase*, ' '),(),()}~~
{('in', ' '),('Alabama'),()}	{(*lowercase*, ' '),('Alabama'),()}
{('in', ' '),(*word*),()}	~~{(*lowercase*, ' '),(*word*),()}~~

(e) {(*lowercase*, ' '),('Alabama'),()} → /[a-z]+ (Alabama)/

Figure 4.10: Regular Expression Inference. (a) The user selects text in a cell. (b) We tokenize selected and surrounding text. For clarity, the figure only includes two neighboring tokens. For each token, we generate a set of matching labels. (c) We enumerate all label sequences matching the text. (d) We then enumerate all candidate *before*, *selection* and *after* combinations. Patterns that do not uniquely match the selected text are filtered (indicated by strike-through). (e) Finally, we construct regular expressions for each candidate pattern.

or selects a current working transform, we assign higher rank to transforms of that type. Second, we consider specification difficulty. We have observed that row and text selection predicates are harder to specify than other parameters. We thus label row and text selections as *hard* and all others as *easy*. We then sort transform types according to the count of *hard* parameters they can accept. Third, we rank transform types based on their corpus frequency, conditioned on their initiating user interaction (e.g., text or column selection). In the case of text selection, we also consider the length of the selected text. If a user selects three or fewer characters, *split* transforms are ranked above *extract* transforms; the opposite is true for longer selections.

We then sort transforms within type. We first sort transforms by frequency of *equivalent* transforms in the corpus. Second, we sort transforms in ascending order using a simple measure of transform *complexity*. Our goal is to preferentially rank simpler transforms because users can evaluate their descriptions more quickly. We define transform complexity as the sum of complexity scores for each parameter. The complexity of a row selection predicate is the number of clauses it contains (e.g., "a=5 and b=6" has complexity 2). The complexity of a regular expression is defined to be the number of tokens (described previously) in its description. All other parameters are given complexity scores of zero.

Finally, we attempt to surface diverse transform types in the final suggestion list. We filter the transforms so that no type accounts for more than 1/3 of the suggestions, unless the transform type matches the working transform or the filter results in fewer suggestions than can appear in the interface.

4.8 Comparative Evaluation with Excel

As an initial evaluation of Wrangler, we conducted a comparative user study with Microsoft Excel. Subjects performed three common data cleaning tasks: value extraction, missing value imputation, and table reshaping. Our goal was to compare task completion times and observe data cleaning strategies. We chose Excel because it is the most popular data manipulation tool and provides an ecologically valid baseline for comparison: all subjects use it regularly and half self-report as experts. Excel also supports our chosen tasks. Neither Potter's Wheel [86] (no support for fill) nor Google Refine [48] (lack of reshaping) support

the full set. In contrast, Excel includes specific tools for each task (text-to-columns, goto-special & pivot tables) in addition to manual editing.

4.8.1 Participants and Methods

We recruited 12 participants, all professional analysts or graduate students who regularly work with data. Subjects rated their prior experience with Excel on a 10-point scale (1 being basic knowledge and 10 being expert); the median score was 5. Participants had never used the Wrangler interface.

We first presented a 10 minute Wrangler tutorial describing how to create, edit, and execute transforms. We then asked subjects to complete three tasks (described below) using both Wrangler and Excel. We randomized the presentation of tasks and tools across subjects. In each task, we asked subjects to transform a data set into a new format, presented to them as a picture of the final data table.

Task 1: Extract Text. In this task, we asked users to extract the number of bedrooms and housing price from housing listings on craigslist. The original data set contained one cell for each listing, with all the information in a text string. The target data set consisted of two columns: one for the number of bedrooms and one for the housing price.

Task 2: Fill Missing Values. We gave users data containing year-by-year agricultural data for three countries. Some of the values in the data set were blank. The target data set contained the same data with all missing values replaced with the closest non-empty value from a previous year.[1]

Task 3: Reshape Table Structure. Users started with three columns of housing data: year, month, and price. The target data set contained the same data formatted as a cross-tab: the data contained one row for each year, with the 12 months as column headers and housing prices as cell values.

When using Excel, we allowed subjects to ask for references to functions they could describe concretely (e.g., we would answer *"how do I split a cell?"* but not *"how do I get the number of bedrooms out?"*). For Wrangler tasks, we did not respond to user inquiries. We permitted a maximum of 10 minutes per task. Each data set had at most 30 rows and

[1] We acknowledge that this is not an ideal cleaning solution for the data, but it nonetheless served as a useful test.

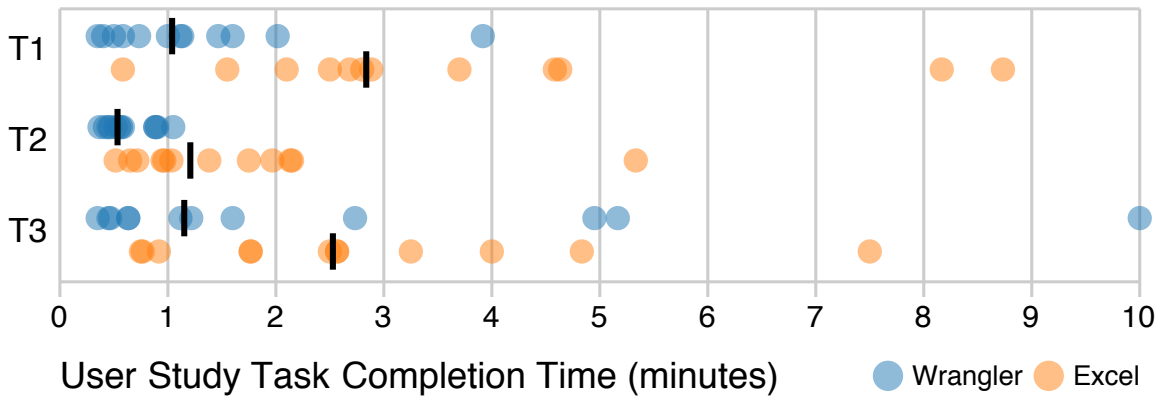


Figure 4.11: Task completion times. Black bars indicate median values. Median Wrangler performance is over twice as fast in all tasks.

4 columns; complete manual manipulation in Excel was easily attainable within the time limits. Afterwards, each user completed a post-study questionnaire.

4.8.2 Wrangler Accelerates Transform Specification

We performed a repeated-measures ANOVA of completion times with task, tool, and Excel novice/expert[2] as independent factors; we log-transformed responses to better approximate a normal distribution. We found a significant main effect of tool ($F_{1,54} = 23.65$, $p < 0.001$), but no main effect of task ($F_{1,54} = 0.01$, $p = 0.943$) or expertise ($F_{1,54} = 0.30$, $p = 0.596$). We found a significant interaction effect of task and expertise ($F_{1,54} = 11.10$, $p < 0.002$) driven by improved performance by experts (regardless of tool) in the reshaping task (T3). No other interactions were significant.

Across all tasks, median performance in Wrangler was over twice as fast as Excel (Fig. 4.11). Users completed the cleaning tasks significantly more quickly with Wrangler than with Excel, and this speed-up benefitted novice and expert Excel users alike. Moreover, the user study tasks involved small data sets amenable to manual manipulation. As data set size grows, we expect the benefits of Wrangler to come into even sharper relief. Of

[2] We divided subjects into "novices" and "experts" according to their median self-reported expertise rating (5).

course, larger data sets might complicate the process of assessing transform effects and so may benefit from additional validation and visualization techniques.

4.8.3 Strategies for Navigating Suggestion Space

When working with Wrangler, users applied different navigation strategies for different tasks. These strategies were largely consistent across users. For text selection, users frequently provided multiple examples. For other operations, users performed an initial selection and then previewed each suggestion. One subject noted, *"I just look at the picture."* Users with a programming background spent time reading transform descriptions, whereas the other users relied almost entirely on the previews. When users did not find a transform among the initial suggestions, they most often filtered the suggestions by selecting a transform type from the menu. If only imperfect matches were found, users then selected the nearest transform and edited its parameters. In other words, users turned to manual parameterization only as a last resort.

Our post-study questionnaire asked users to rate automated suggestions, visual previews, and direct editing of transforms on a scale from 1 (not useful) to 5 (most useful). We performed an ANOVA and found a significant difference among the ratings ($F_{2,33} = 17.33$, $p < 0.001$). Users rated previews ($\mu = 4.8$) and suggestions ($\mu = 4.3$) significantly more useful than direct editing ($\mu = 2.5$) ($p < 0.001$ in both cases by Tukey's HSD). Users' preference for suggestions and previews over direct editing provides evidence that these novel user interface features have merit.

Users' navigation strategies worked well when they understood the nature of the desired transform, even if they did not know how to specify it. However, we found that users of both tools experienced difficulty when they lacked a conceptual model of the transform. For instance, Task 3 exhibited an uneven distribution of completion times; 7 of the 10 fastest times and 3 of the 4 slowest times were in Wrangler. Wrangler does not provide the recourse of manual editing, hence users who got stuck fared slightly better in Excel. However, those familiar with pivot operations in Excel uniformly performed the task more quickly with Wrangler.

We also observed one recurring pitfall: a few users got stuck in a "cul-de-sac" of suggestion space by incorrectly filtering (e.g., by selecting a specific transform type from the menu). When this happened, some users kept searching and refining only these filtered transforms. By design, Wrangler does not afford users the same flexibility to layout data as in Excel; since users cannot perform arbitrary editing in Wrangler, the recourse is less obvious when they get stuck. This pitfall was most common in Task 3, where a user might mistakenly filter all but *fold* operations when an *unfold* operation was needed. One solution may be to suggest non-matching transforms related to the selected transform type, in effect treating filtering criteria as guidelines rather than strict rules.

4.9 Proactive Wrangler

We addressed some of shortcomings of Wrangler in its successor, Proactive Wrangler [35]. We redesigned the Wrangler user interface to better surface available transforms and their parameters. The tool bar along the top of the user interface makes all transforms immediately visible, and the parameters for a current transform appear directly below. As users preview suggested transforms, the complete parameterization is now visible in the tool bar. We apply linked highlighting between parameters in natural language descriptions and those in the tool bar.

To help analysts formulate multi-step transformations, we extended Wrangler to provide proactive suggestions in addition to "reactive" suggestions initiated by user actions. Our proactive suggestions are intended to lead users towards effective cleaning strategies and facilitate the specification of complex reshaping operations. We designed a model to proactively suggest data transforms which map input data to a relational format expected by analysis tools. To guide search through the space of transforms, we propose a metric that scores tables according to type homogeneity, sparsity and the presence of delimiters. When compared to "ideal" hand-crafted transformations, our model suggests over half of the needed steps; in these cases the top-ranked suggestion is preferred 77% of the time. User study results indicate that suggestions produced by our model can assist analysts' transformation tasks, but that users do not always value proactive assistance, instead preferring to maintain the initiative.

4.10 Conclusion and Future Work

This chapter introduced Wrangler, an interface and underlying language for data transformation. The system provides a mixed-initiative interface that maps user interactions to suggested data transforms and presents natural language descriptions and visual transform previews to help assess each suggestion. With this set of techniques, we find that users can rapidly navigate to a desired transform.

Our user study demonstrates that novice Wrangler users can perform data cleaning tasks significantly faster than in Excel, an effect shared across both novice and expert Excel users. We found that users are comfortable switching navigation strategies in Wrangler to suit a specific task, but can sometimes get stuck—in either tool—if they are unfamiliar with the available transforms. Future work should help users form data cleaning strategies, perhaps through improved tutorials.

Chapter 5

Debugging Data Transformation Scripts

Analysts commonly author and refine transformation scripts on a sample of a larger data set before applying the script to remaining data. In many cases, these transformation scripts, which may take hours or days to run on the entire data set, fail or output unexpected results. Identifying exceptions and their causes in the output is time-consuming and tedious for large data sets. In this chapter we contribute two novel methods to aid rapid debugging of Wrangler transformation scripts run at scale. *Surprise-based anomaly detection* applies a model to classify output records as exceptions. We empirically evaluate this method on a corpus of log files. *Rule-based transform disambiguation* generates example records to help analysts refine transformation scripts before applying them. We discuss the complexity and challenges of using this method.

5.1 Introduction

When working with large data sets, an analyst typically authors a script, applies it to a subset of the data, evaluates the output of the script, and then refines the scripts as necessary. By working on a small subset, the analyst can quickly iterate on the script until it produces the desired output over the sample. Once the script is satisfactory, the analyst then applies the script to the entire data set.

However, in many cases, the transformation may fail during execution or complete with results that differ from what the script's author intended. Such errors may occur because

input files can contain heterogeneous record structures or other exceptions not present in the original sample. For instance, a web server log file may contain entries representing user account login events as well as user transactions. A script that was tested on a sample only containing login events would likely break when applied to records containing user transactions.

Inspecting a script's output to determine if an error has occurred can be difficult for large data sets. Manually inspecting a data set is cumbersome for a data set containing even thousands of records and completely impractical for those containing millions or billions of records. Authoring constraints or other auditing methods can help catch certain types of errors, but these constraints can be difficult for non-programmers to create. Summary visualizations can help analysts spot erroneous data, but often require that the data is already in a suitable format. Automated routines for anomaly detection also often require that data is well-formatted or already organized in columns with homogeneous data types.

Reasoning about the transformation script itself can often help uncover flaws in logic or other potential problems. For instance, programmers may uncover edge cases that are unaccounted for by the script or realize a script does not generalize to slightly different record types. However, it can be difficult, even for expert programmers, to anticipate all of the potential edge cases necessary to modify a script. Instead, it is common to incrementally refine a script to address potential issues as they arise in the output. However, this practice grows costly as script running times increase with the size of the data.

In this chapter, we contribute two methods to address these issues and aid rapid debugging of transformation scripts. *Rule-based transform disambiguation* provides a mechanism for generating examples to help analysts refine transformations. We evaluate the complexity of this technique and discuss issues applying it. *Surprise-based anomaly detection* leverages a model to classify output records as exceptions. We evaluate this model on a corpus of a seven log files containing a total of approximately 16,000 records. *Rule-based transform disambiguation* can provide synthetic examples without the cost of applying a transformation to any of the data. *Surprise-based anomaly detection* identifies potential anomalies within existing records, but requires applying transformations to each of those records.

Next, we describe a motivating example and characterize the frequency of the above

Sample

Gladiator (2000)
Amadeus (1984)
Full Metal Jacket (1987)

Remainder

2001: A Space Odyssey (1968)
Braveheart (1995)

Figure 5.1: Data Extraction Error. The analyst applies a regular expression matching four digit numbers to extract the release year for a set of movies. The expression extracts the correct data from the sample, but extracts the wrong data from the first record in the remaining data.

issues in a corpus of log files. We then describe and evaluate both methods. We conclude by discussing limitations of our work and opportunities for future research.

5.2 Motivation

To motivate our techniques for debugging transformation scripts, we provide an example of a typical Wrangler workflow. We then discuss the types of errors that can arise during this workflow.

5.2.1 Motivating Example

Consider an analyst working with movie data containing the title and release year for each movie (Fig. 5.1). To keep the example simple, the sample contains three records, and the entire data set contains five records. In practice, a sample may contain thousands or tens of thousands of rows and the entire data set may contain millions or more records.

The analyst uses Wrangler to build a transformation script on the sample of movies. The analyst loads a sample of the data set into Wrangler (3 rows) and attempts to extract

the release year for each movie into its own column. She selects the year in each of the records and Wrangler generates candidate transforms. For instance, Wrangler suggests a transform that selects the first number in each row and another suggestion that selects numbers between parentheses in each row.

By using the visual preview, she confirms that the former transform selects the appropriate text for each movie in the sample and decides to execute it on the sample. Satisfied with the script, she exports generated Python code and applies it to the entire data set. After running the script, she visually inspects the output to confirm it worked on the remaining records. She confirms that the script extracted 4 digit numbers for each movie and assumes, incorrectly, that the output is correct. She exports this data into another tool and uses the incorrect data in a subsequent analysis.

5.2.2 Types of Errors

The example scenario demonstrates two common issues — *transform ambiguity* and *surprising features* — that occur when transforming data sets and which we will address throughout the rest of the chapter.

Transform Ambiguity. Multiple transforms may have the same effect when applied to a set of data. An analyst relying solely on visual previews can not distinguish between these transforms. However, when the transforms are run on new or updated data, they may have different results and the analyst may choose the wrong one to apply. For instance, in the scenario above, the transform that extracted numbers between parentheses would have produced desired results on the entire data set.

Surprising Features. Second, the entire data set may contain records with features not present in any of the records in the transformed sample. For instance, in the scenario above, "2001: A Space Odyssey", contains an integer in the title of the movie. This feature is not present in any of the movies in the sample. Features not present in the sample can often cause unintended behavior when applying a transform.

For the remainder of the chapter, we focus on transforms that involve complex selection criteria, such as regular expressions. We focus on this set of transforms because they

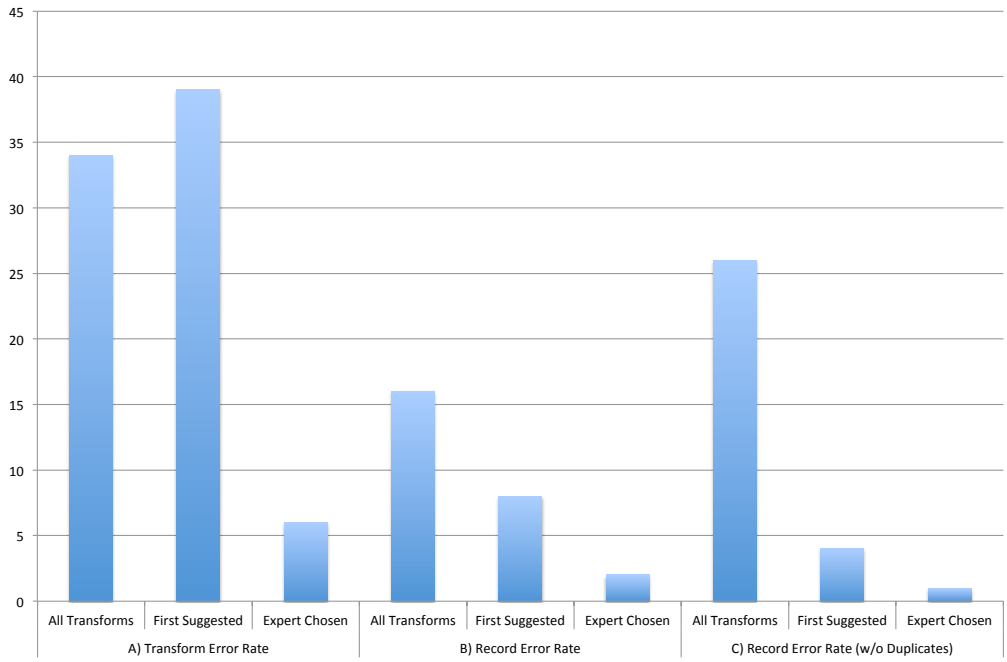


Figure 5.2: Transform Error Rates. (A). The percentage of transforms that result in at least one error. (B) The percentage of all records that parse incorrectly. (C) The percentage of all records that parse incorrectly after removing duplicate records.

are the most common wrangling transformation in our usage data, are difficult for non-programmers to reason about, and users rely heavily on visual previews to assess their effects. In particular, we focus on text extraction transforms, though the discussion has a straightforward generalization to other text manipulation transforms such as text splitting and text deletion.

5.2.3 Transform Error Rates

We empirically evaluated how often parsing errors occurred with text extraction transforms in a corpus of 7 log files containing a total of approximately 16,000 records. The corpus contained a mixture of log files, including web server, operating system, printer, sensor, network, and package manager logs. For each log file, we sampled the first 25 rows. We

then used Wrangler to extract each field of the sample, including dates, entity names, error codes and locations, into their own columns. In total, we extracted 49 fields. For each extracted field, Wrangler's suggestion engine (described in 4.7.2), enumerates a set of transforms that parsed the sample correctly. For each field, this set consisted of an average of 18 transforms.

We then ran this set of suggested transforms over the entire data set and compared their output against a reference data set. For each of the 49 extracted fields, Wrangler always suggested at least one transform that matched the entire data set.

However, many of the enumerated transforms failed to produce the correct output. We considered three subsets of the enumerated transforms throughout the rest of the chapter. T_{ALL}, is the set of all enumerated transforms. The average error rates here correspond roughly to an analyst who chooses randomly from the set of suggested transforms. T_{FIRST}, is the set of the highest-ranked transforms in the ranked-order list of Wrangler suggestions for each field. This set corresponds to the user who picks the first suggested transform that produces the correct output on the sample. Finally, we manually picked the transform for each field that most closely matched (according to our judgement) the extraction criteria used to generate the reference data set, T_{EXPERT}. This set corresponds to an expert user who considers not only the transform's output, but also reasons about it's description.

We calculated error rates (Figure 5.2) by comparing the output of each transform to the reference output. 34% of transforms in T_{ALL}, 39% of T_{FIRST} and 6% of T_{EXPERT} resulted in at least one error when run on the entire data set. One possible explanation for the higher error rate of T_{FIRST} is that the Wrangler suggestion engine preferentially ranks extraction rules that use simple string or positional matches instead of regular expressions. These rules tended to be more brittle when generalizing to the larger data sets in our samples.

We also computed the total number of parsed fields that resulted in an error for each transform. 16% of the fields for T_{ALL}, 8% for T_{FIRST} and 2% for T_{EXPERT} differed from the reference value. When we removed duplicate records from the log files, the error rates were 26%, 4% and 1% respectively.

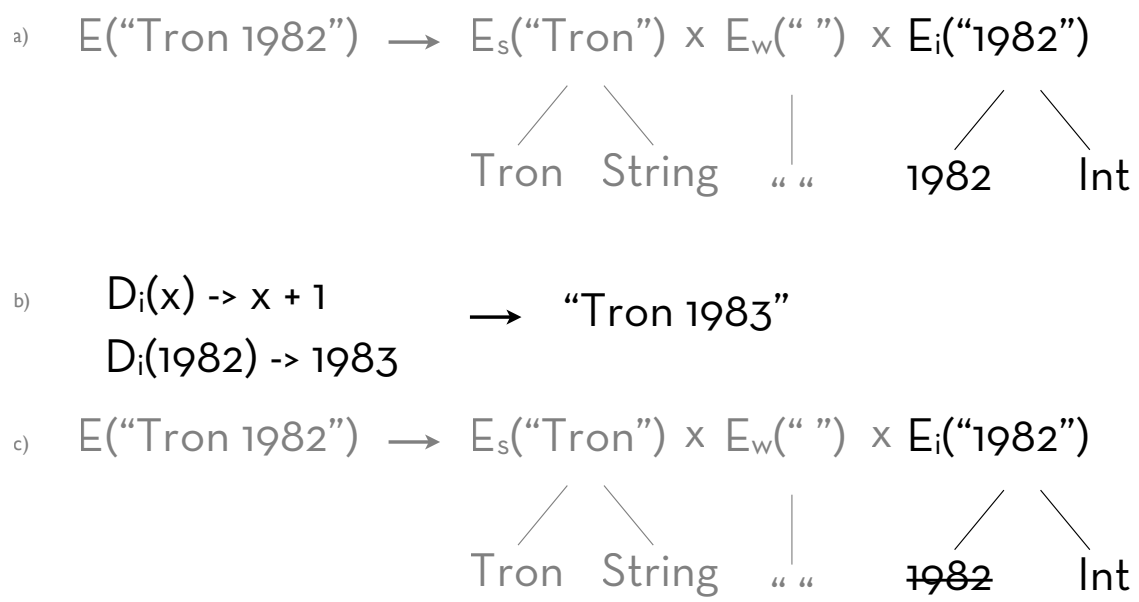


Figure 5.3: *Rule-based transform disambiguation*. a) A sample enumeration of possible rules for matching tokens in text. b) An example disambiguation function to distingiush between any integer and a constant integer. c) The pruned tree of enumerations after user confirmation.

5.3 *Rule-based transform disambiguation*

Rule-based transform disambiguation takes as input a set of text extraction transforms generated by the Wrangler inference engine and a record from the data set, and constructs a set of example records. The example records are constructed such that if a user applies the set of input transforms on the set of example records, at most one transform will generate the intended output for all records. This constraint on the example records enables an end-user to prune the set of transforms through visual inspection.

5.3.1 Method

Our method for generating a disambiguating sample consists of four phases. In the first phase, we tokenize the input record. Next, for each token we enumerate a space of possible regular expressions that match the given input record using enumeration rules. In the third

phase, we apply a set of disambiguation functions to the input record. We use the results of these functions to construct the disambiguating sample in the fourth phase. We discuss each part of this process in more detail below.

Tokenization. We tokenize an input record into a set of disjoint token sets using the same method discussed in Section 4.7.2. This tokenization supports integer, whitespace, string and one-character symbol tokens. As an example, consider the text "Tron 1983" in Figure 5.3. Tokenizing this string results in three tokens: *string*, *whitespace*, *integer*. In this section, we restrict discussion to the tokenization scheme used in the Wrangler inference engine (Section 4.7.2). In subsection 5.3.2, we discuss extensions to this tokenization schema to enable support for richer data types.

Enumeration Rules. Recall from Section 4.7.2 that Wrangler enumerates a space of regular expressions from examples using a set of enumeration rules. Each enumeration rule accepts a token (typed substring) as input, and outputs a set of possible regular expressions for matching that string. For instance, consider the example in Figure 5.3. We show a possible enumeration of rules for matching the text "Tron 1982". After tokenizing the string in to *string*, *whitespace*, *integer*, we apply the appropriate enumeration rules for each token. When applied to the token "1983", the rule generates a set of regular expressions (regex) including a regex matching all integers and a regex matching the literal "1983". By taking the cross product of the output of each enumeration rule, and concatenating regular expressions, we generate a set of candidate regular expressions.

Disambiguation Functions. To generate disambiguating examples, we annotate each enumeration rule with a set of corresponding disambiguation functions. A disambiguation function accepts a token and generates a disambiguation token. These set of generated disambiguation tokens need to satisfy that no two distinct regular expressions generated by the enumeration rule agree on the set of all tokens.

Figure 5.3 shows a disambiguation function for a simple enumeration rule for integers. The enumeration rule generate two regular expressions to match integers: a constant matching the input value, and a regular expression matching all integers. The disambiguation function increments the input integer. The constant regular expression does not match the new integer, and the regular expression matching all integers does, so this is a valid disambiguation function.

(string)	
Enumeration Rule	{string ($[A-Za-z]+$), fixed-length string ($[A-Za-z]\{k\}$), constant string (k)}
Disambiguation Functions	$\{replaceFirstCharacter, appendCharacter\}$
Example Input Token	Tron
Disambiguating Examples	{Uron, Trona}
(integer)	
Enumeration Rule	{integer ($\backslash d+$), fixed-length integer ($\backslash d+\{k\}$),constant integer (k)}
Disambiguation Functions	$\{incrementInteger, appendDigit\}$
Example Input Token	1982
Disambiguating Examples	{1983, 19820}
(float)	
Enumeration Rule	{<integerEnumerationRules>.<integerEnumerationRules>}
Disambiguation Functions	<integerDisambiguation> $\times$ <integerDisambiguation>
Example Input Token	5.67
Disambiguating Examples	{5.68, 5.670, 6.67, 50.67}
(Date (month/year))	
Enumeration Rule	{<fixed-lengthIntegerEnumerationRules>/<fixed-lengthIntegerEnumeration>}
Disambiguation Functions	{<fixed-lengthIntegerDisambiguation>$\times$<fixed-lengthIntegerDisambiguation>}
Example Input Token	11/1982
Disambiguating Examples	{12/1982, 11/1983}
(State Name)	
Enumeration Rule	{any state ($\bigcup_{States} StateName$)/constant value (k)}
Disambiguation Functions	$\{distinctDictionaryValue\}$
Example Input Token	Alabama
Disambiguating Examples	{Alaska}

Figure 5.4: Disambiguation Functions. Disambiguation functions corresponding to enumeration rules for integers, strings, floats and state names. Floats are an example of a compound type and state names are an example of a dictionary type. For each rule, we show candidate patterns, a set of disambiguating functions, an example input token, and the modified examples used for disambiguation.

Disambiguation Sample. Now if we take the cross product of the sets of disambiguation tokens and concatenate the tokens in each set, we get a set of strings. Given the input record, for each of these strings, we can generate a new record by replacing the original matched string with this new string. Doing so results in a set of new records, which we can present to the user. By construction, none of the candidate regular expressions will agree on what they match in all records of the sample. We can then prompt the user to choose which of the regular expressions has the intended result on this sample. If no records have the intended result, then the set of candidate transformations is inadequate. We do not address this case here, as that is a deficiency in the suggestion algorithm itself.

5.3.2 Extension to other Types

The tokenization scheme above only recognizes strings, integers, symbols and whitespace as tokens. To support richer data types we can extend the set of tokens to recognize additional common patterns such as *floats*, *dates*, or *ip addresses*. Note that adding these patterns introduces a Token Ambiguity Problem [86, 109]: a given record may have multiple valid tokenizations. For instance, the record "3.14", may parse as *integer* . *integer* or as *float*. There are numerous approaches for choosing a tokenization, including information-theoretic approaches [86] and statistical approaches [109]. We employ a simple heuristic approach: choosing the tokenization containing the fewest number of tokens. However, the rest of the discussion applies to any tokenization method.

Given a tokenization, we describe general approaches for building disambiguation functions for two classes of types: dictionary types and compound types.

Dictionary Types

A dictionary type is a type that accepts strings if and only if they are contained within a dictionary. Example of dictionary types include country names, chemical elements or movie ratings. A generic disambiguation function for a dictionary swaps the token with another token in the dictionary to decide if the token needs to be exactly that token or just any token in the dictionary. Figure 5.4 shows disambiguating functions for a U.S. state name dictionary.

Compound Types

A compound type is a type that is composed of two or more subtypes. Examples include dates (e.g., day/month/year) or locations (city name, state name). To construct disambiguating functions for compound types, we can simply take the cross product of corresponding disambiguating functions for subtypes. Figure 5.3 shows examples of such functions for the compound types float and date.

Our method above extends to handle these other token classes. For any token class, the system must register both enumeration rules and disambiguation functions for that class. Our algorithm then treats these classes as it would any other token class.

5.3.3 Method Complexity

The method above requires authoring disambiguation functions, applying these functions to an input record and evaluating the output of these functions. We discuss the complexity of these three components below.

Authoring Complexity

Applying *Rule-based transform disambiguation* requires writing a disambiguation function for each token enumeration rule. The number of rules is therefore dependent on the number of enumeration rules and the number of functions per enumeration rule. The number of rules is dependent on the number of distinct token classes, C present in the suggestion engine. A rule that produces k possible outputs will require a minimum of $log_2(k)$ disambiguation functions to produce enough distinguishing examples for the general case. Together, this method requires writing $C * log_2(k)$ disambiguating functions.

Computational Complexity

All of the disambiguation functions in our discussion require constant time and space to evaluate. Since there $C * log_2(k)$ disambiguation functions, this method requires $O(C * log_2(k))$ running-time and space to complete.

Labeling Effort

The biggest cost in this loop is the amount of time it takes the end-user to evaluate the set of transforms on the generated sample. By definition, each record should eliminate at least one transform as a candidate. In the worst case, the user will therefore need to evaluate $|T| - 1$ generated examples, for a set of transforms T.

5.3.4 Comparison to Alternative Approaches

Of course, there are other methods to disambiguate candidate transforms. Wrangler users could read and interpret the natural language descriptions to reason about their effects. However, in user studies of the tool, we found that non-programmers often preferred or relied on visual previews.

Second, users could provide examples on their own. Again, for this method to work efficiently, users would need to have an understanding of how transformations differ from each other and what types of examples would exercise their differences. Also, it can be tedious to create these examples even if the model is understood.

Another method to find disambiguating examples is to apply all transformations in parallel on a large data set, and present output records that are not in agreement. Unlike *Rule-based transform disambiguation*, this method will discover disambiguating examples that actually exist in the data set. This method suffers from two drawbacks compared to *Rule-based transform disambiguation*. First, it can have significantly increased computation costs to apply the set of transforms on records in the data set. Second, there is no guarantee that the disambiguating example is available in the data set yet. For instance, after the transform is authored, the data set may update and then contain new records containing the example. With *Rule-based transform disambiguation*, the disambiguation will terminate after a fixed set of records.

5.3.5 Syntactic vs. Semantic Correctness

One possible pitfall of this method of disambiguation is the tension between syntactic and semantic correctness. For instance, in Figure 5.3, "Tron 1983" is syntactically correct for

		Matched	Prefix	Suffix
Record	Cars 2 (2011)	i	sw	w(i)
Sample	Jerry Maguire (1996)	i	swsw(	)
	Forrest Gump (1994)	i	swsw(	)

Figure 5.5: Tokenization. We tokenize the text matched match by the regular expression, as well as a k-token (k=5) window prefix and suffix of the matched-text. Consecutive characters are mapped to a string token (s), integer characters are mapped to an integer token (i), and all other characters map to themselves. We use (w) to denote whitespace.

the transformation rule, but the data in the example is semantically incorrect (Tron was made in 1982, not 1983). Other manipulations, such as truncating strings or changing whitespace may also cause even more confusion. It is vital that users understand what they are verifying. User interfaces leveraging this technique must make this distinction clear to the end-user. When we consider the analytic life-cycle more broadly, some of the aspects in this stage can be dealt with in subsequent stages of analysis. For instance, we can delay semantic assessment to the profiling stage. User studies should address how well users understand this staging and future work should address these limitations.

5.4 *Surprise-based anomaly detection*

In *Surprise-based anomaly detection*, we train a model to classify records as exceptional with respect to a given transform and sample data set. We assume the user has verified that the transform behaved correctly on a sample of data before applying the transform to the entire data set. Since every record in the sample is verified to be correct, our model will use features which identify characteristics of records that differ form features anywhere in the sample.

Let $tokenSet(r)$ be the set of distinct tokens appearing in a record, r.
Let $count(t,r)$ be the number of times a token, t, appears in a record r.

$$\textbf{UnseenToken}(r) = I(tokenSet(r) \nsubseteq \bigcup_{s \in S} tokenSet(s))$$

$$\textbf{MissingToken}(r) = I(\bigcup_{s \in S} tokenSet(s) \nsubseteq tokenSet(r))$$

$$counts(t,S) = \bigcup_{s \in S} count(t,s)$$

$$tokenSurprise(t) = \begin{cases} 1 - normalizedEntropy(S) & \text{if } count(t,r) \notin counts(t,S) \\ 0 & \text{otherwise} \end{cases}$$

$$\textbf{TokenCountSurprise}(r) = max_{t \in tokenSet(r)} tokenSurprise(t)$$

Figure 5.6: Feature Definitions. We use three features in *Surprise-based anomaly detection*. The *UnseenToken* feature is an indicator variable indicating that a record contains a token previously not seen in the sample before. The *MissingToken* feature is an indicator variable indicating that a record does not contain a token seen in all of the sample records. The third feature, *TokenCountSurprise*, is a measure of how much the frequency of a token differs from the frequency with which it appears in any sample records.

5.4.1 Model Features

To construct features, we first apply the candidate regex on each record in the data set. We then consider three substrings of the record: the text matched by the regex and the prefix and suffix of the matched text. We then tokenize each of these substrings, preserving a k-token window for the prefix and suffix. Figure 5.5 shows tokenizations of the three regions for three records.

Then for each record, we count the number of times each token type appears in the prefix, suffix and matched text for each record. For instance, the three-token prefix of the first record in (Figure 5.5) contains one string token, one whitespace token, and one parentheses token.

Using these token counts we then construct three types of features (Figure 5.6). The *UnseenToken* feature is an indicator variable indicating that a record contains a token previously not seen in the sample before. Similarly, the *MissingToken* feature is an indicator variable indicating that a record does not contain a token seen in all of the sample records.

The third feature, *TokenCountSurprise*, is a measure of how much the frequency of a token differs from the frequency with which it appears in any sample records. In particular,

Substring	Feature Type	Value	Explanation
Matched	UnseenToken	0	No unseen tokens
	MissingToken	0	No missing tokens
	TokenCountSurprise	0	No surprising counts
Prefix	UnseenToken	0	No unseen tokens
	MissingToken	1	Missing '('
	TokenCountSurprise	1	's' appears once instead of twice
Suffix	UnseenToken	1	Contains 'w', 'i'
	MissingToken	0	No unseen tokens
	TokenCountSurprise	0	No unseen tokens

Figure 5.7: Example of Features. We show computed features for the record in Figure 5.5, relative to the sample of two other records. The *selection+context* model uses all 9 features, while the *selection* model only uses the first three features for the "Matched" text.

for a given token that appears at least once in any record of the sample, we compute the entropy over the frequency with which it appears in each record of the sample. For a new record, we sets its *Deviance Entropy* for a given a token to 0 if the frequency matches the frequency for any record in the sample, and to (1 - the normalized entropy) otherwise. We then find the maximum of all such entropies for each distinct token in the record.

5.4.2 Statistical Classifiers

We trained three classifiers — logistic regression, support vector machine and random forest classifiers — over labeled training data. Below, we empirically evaluate their relative effectiveness over our log-file corpus. For each classifier, we evaluated the classifier on two sets of the features described above: *selection* features and *selection+context* features (Figure 5.7). The *selection* features only consider the three features above on the text matched by the regular expression. The *selection+context* features consider the matched text as well as the prefix and suffix of the matched text, resulting in all 9 features.

5.4.3 Model Results

We tested these classifiers using *leave-one-out* cross-validation on our corpus of 7 data sets. For each data set, we trained the classifiers against the other six data sets. We then labeled

Classifier	Transform Set	Accuracy	Precision	Recall	F-score
Random Forest	All	.99	.97	.99	.98
	First	.99	.77	.89	.83
	Expert	.99	.32	.96	.48
Logistic Regression	All	.96	.89	.99	.93
	First	.96	.48	.89	.63
	Expert	.96	.12	.96	.22
SVM	All	.75	.51	.99	.67
	First	.68	.11	1.0	.21
	Expert	.67	.02	1.0	.04

Table 5.1: Model Results. Precision, recall, accuracy, and F-score of models using features of selected text and surrounding text. We show each metric for each set of transforms and three classifiers.

each record of output for each field in the validation data set as correct or as an exception and compared these labels to the corresponding reference data sets.

Figure 5.1 shows the model accuracy, precision, recall and f-score for all three classifiers with each set of features as well as the results of the baseline model. Here we discuss results for our chosen model, the random forest model using *selection+context* features, as it performed the best (as measured by f-score) on all three data sets. In the case of SVM, we only report results for the kernel (linear) that resulted in the best performance.

Our model has an accuracy of 99% (T_{ALL}), 98% (T_{FIRST}), and 99% (T_{EXPERT}) for the three sets of transforms. This model has higher accuracy than the logistic regresssion model and SVM model in all cases. Our model has precision of 96% for T_{ALL}, 77% for T_{FIRST} and 33% for T_{EXPERT}. Again, this model outperforms the other three models in all cases. Our model has recall of 98%, 89%, and 96% for those sets respectively. Again, it has at least as good recall as the linear model in all three cases, but lower recall than the SVM model. The model has an F-score of .97, .87 and .48, again better than the other two models in all three cases.

The *context+selection* model, outperformed the *selection* model for all metrics. To compute features for the prefix and suffix, it is not sufficient to simply store the output matches. We must also store the indices of the match as well as the regex or the prefix suffix at the time of computation. So although the *selection* model does not perform as

well, it has the advantage that it can be applied to output data after it has been processed. The *selection* model might therefore be useful if it is impractical to alter computation.

All of the models we considered performed best on T_{ALL} and worst for T_{EXPERT} on all metrics. Recall that T_{ALL} and T_{FIRST} attempt to approximate the choices a novice user would make in Wrangler. Our model, therefore, seems more applicable for novice users than expert users. The lower precision for T_{EXPERT} is not as big a problem as it would be for the other transform sets as the error rate is much lower for experts. Overall, our model seems to perform well, given that recall is high and precision is reasonable in most cases.

5.5 Extensions to Wrangler Interface

Both *Surprise-based anomaly detection* and *Rule-based transform disambiguation* leverage human input to aid debugging. We modify the Wrangler interface (Chapter 4) to enable end-users to provide input to these algorithms and display their results. Recall that the original Wrangler interface presented a sample consisting of the first X rows of data, sorted in their original order. We still present this sample to the user when the user initially loads data, as the top of file often contains useful information such as schema information embedded in header rows. However, as a user begins transforming data, Wrangler can augment this sample with two sets of records: synthetic records automatically created by Wrangler and anomalous records identified by Wrangler and chosen by the user.

5.5.1 Synthetic Example Records

Recall in the original interface, users disambiguate suggested extraction transforms by providing more examples of data to be extracted. In the modified interface, users can still provide more examples. However, when the visible sample does not contain any examples that would disambiguate the transforms, the interface now displays a set of synthetic examples (Figure 5.8), generated by *Rule-based transform disambiguation*, that are sufficient to disambiguate the suggested transformations. These records are placed at the top of the sample. Since, most of these records are unlikely to be contained within the original data set, we mark these these records as synthetic and highlight them to avoid confusing the

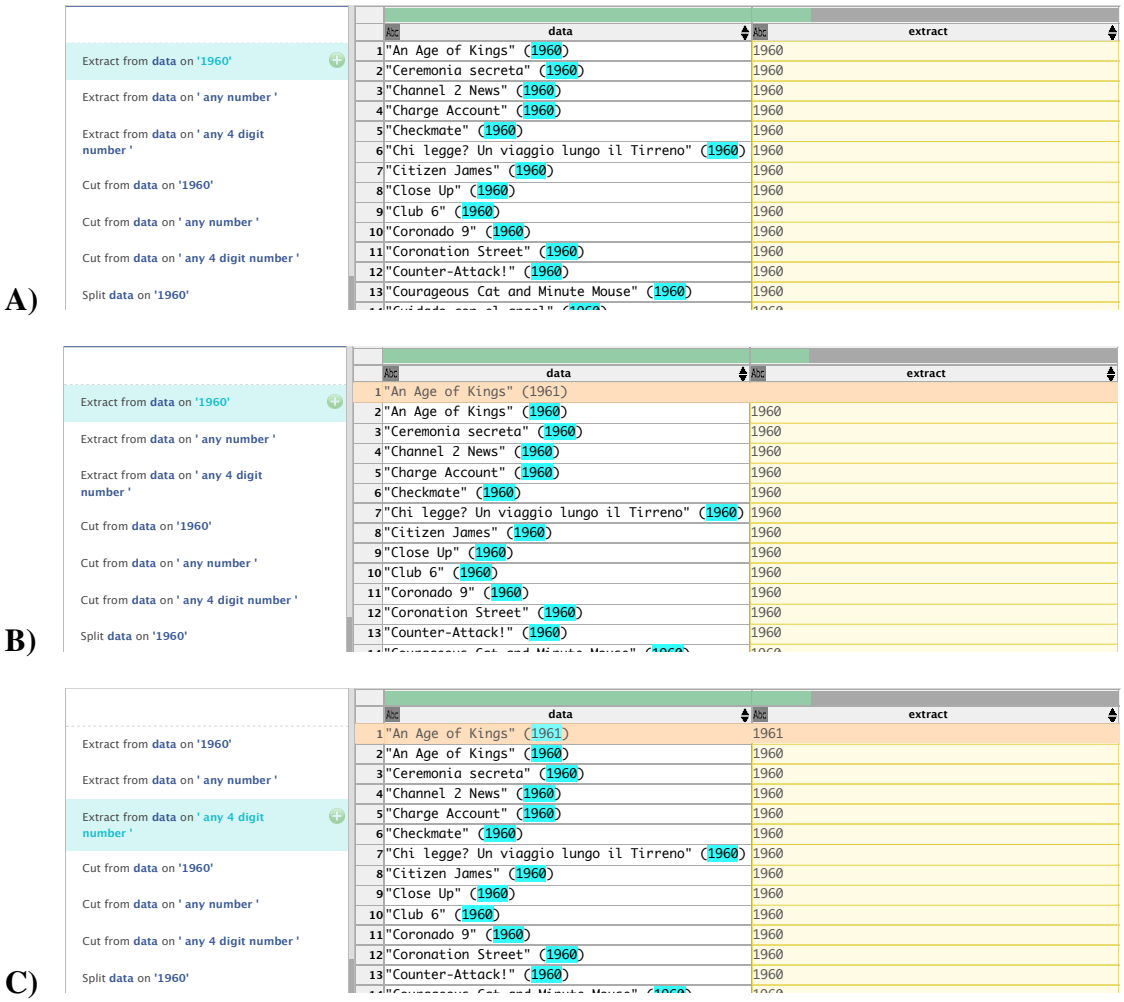


Figure 5.8: Synthetic Examples. A). After selecting the text "1960" in rows 1 and 2, the original Wrangler suggests three extraction transformations. With the current sample of records, the first suggestion can not be distinguished from the second suggestion. B). In the updated interface, Wrangler inserts an additional synthetic record at the top of the sample, for the user to disambiguate the first suggestion from the second suggestion. C). After mousing over the third suggestion, the user finds the desired transformation.

user. The user can interact with this set of records in the same way they would any other record in the sample. Users have the option to toggle this feature in application settings.

5.5.2 Discovering and Resolving Anomalies

After a user executes an extraction transform in the original interface, Wrangler adds this transform to an interactive Transform History. In the new interface, we now use *Surprise-based anomaly detection* to identify potentially anomalous output records. We run *Surprise-based anomaly detection* on any records present in the sample that the user has not yet visually inspected (recall a sample may contains tens of thousands of records, and the user may only inspect a small subset of those records). If *Surprise-based anomaly detection* reports any potential anomalies, we annotate the transform in the transform history with a red flag.

When a user clicks on the red flag, we display a new sample of records in a debugging interface (Figure 5.9). The debugging interface shows a sample of records flagged as anomalies by *Surprise-based anomaly detection*. The user can then perform three actions on these records. First, a user can choose to move a record to the top of the *active sample*. Second, a user can mark an anomaly as ignored. We set the opacity of ignored records to 0.5 to fade them out. The user can unignore an ignored record. Third, a user can toggle the label of a record. *Surprise-based anomaly detection* uses these labels in the next iteration of classification. Third, When the user returns to editing mode, such records will be at the top of the active sample. The user can choose to edit the transform or can perform additional operations to patch the records.

5.6 Conclusion

In this chapter, we contributed two methods to aid users debugging transformation scripts. *Rule-based transform disambiguation* provided a mechanism for generating examples to help analysts refine transformations. In the worst case, an end-user will need to evaluate $|T| - 1$ synthetic example records to disambiguate a set of transforms T. *Surprise-based*

A)

Suggestions

rows: 35806 prev next Move to Top Ignore Mark OK

	data	extract
1	1. maj 1960 (1960)	1960
2	1001 crtez (1960)	1001
3	16 ottobre 1943 (1960)	1943
4	2000 naftes ki ena koritsi (1960)	2000
5	9. kvetna 1960 (1960)	1960
6	The Good, Bright Days (1919-1927) (1960)	1919
7	Beograd 1960 (1960)	1960
8	Teknillinen korkeakoulu 1959 (1960)	1959
9	Sunshine and Eclipse (1927-1934) (1960)	1927
10	Project 1251 (1960)	1251
11	Praznik rada 1960 godine (1960)	1960
12	Natal 1960 (1960)	1960
13	Twilight of an Era: 1934-1939 (1960)	1934
14	Joulukuun kuudes 1960 (1960)	1960
15	"Call Oxbridge 2000" (1961)	2000

B)

Suggestions

rows: 35806 prev next

	data	extract
1	1. maj 1960 (1960)	1960
2	1001 crtez (1960)	1960
3	16 ottobre 1943 (1960)	1960
4	"An Age of Kings" (1960)	1960
5	"Ceremonia secreta" (1960)	1960
6	"Channel 2 News" (1960)	1960
7	"Charge Account" (1960)	1960
8	"Checkmate" (1960)	1960
9	"Chi legge? Un viaggio lungo il Tirreno" (1960)	1960
10	"Citizen James" (1960)	1960
11	"Close Up" (1960)	1960
12	"Club 6" (1960)	1960
13	"Coronado 9" (1960)	1960
14	"Coronation Street" (1960)	1960
15	"Counter-Attack!" (1960)	1960

C)

Suggestions

Extract from data between '(' and ')'

Extract from data between ' (' and ')'

Extract from data between ' (' and ')'

Cut from data between '(' and ')'

Cut from data between ' (' and ')'

Cut from data between ' (' and ')'

Split data between '(' and ')'

rows: 35806 prev next

	data	extract
1	1. maj 1960 (1960)	1960
2	1001 crtez (1960)	1960
3	16 ottobre 1943 (1960)	1960
4	"An Age of Kings" (1960)	1960
5	"Ceremonia secreta" (1960)	1960
6	"Channel 2 News" (1960)	1960
7	"Charge Account" (1960)	1960
8	"Checkmate" (1960)	1960
9	"Chi legge? Un viaggio lungo il Tirreno" (1960)	1960
10	"Citizen James" (1960)	1960
11	"Close Up" (1960)	1960
12	"Club 6" (1960)	1960
13	"Coronado 9" (1960)	1960
14	"Coronation Street" (1960)	1960
15	"Counter-Attack!" (1960)	1960

Figure 5.9: Debugging Interface. A). In the debugging interface, Wrangler displays examples of potentially anomalous records. Here, the user has selected the first three anomalous records. The user can choose to move these records to the top of the sample, ignore the records, or relabel them as OK. B). After choosing to add these records to top of the sample, the user returns to the editing interface. The new sample contains these records on the first page of visibility. C). The user can edit the extract transform to account for the anomalous records.

anomaly detection leveraged a random forest classifier to classify output records as exceptions. We evaluated this model on a corpus of log files, and compared it to two other models. The random forest model had higher precision than the other models with comparable recall. All models performed worst on a set of expert chosen transformations, indicating the technique might be more applicable for novice users than expert users. We also discussed modifications to the Wrangler interface (Chapter 4), enabling end-users to provide input to these methods and display their results. Next, we compare *Rule-based transform disambiguation* and *Surprise-based anomaly detection* and discuss limitations of both algorithms and discuss future work for improving the Wrangler interface.

5.6.1 Preventative and Iterative Debugging Strategies

Rule-based transform disambiguation and *Surprise-based anomaly detection* are complementary and can be used in conjunction with each other. However, both methods compete for user attention and potentially for computational resources. With *Rule-based transform disambiguation*, the user engages in a preventative strategy of refining a transformation. With *Surprise-based anomaly detection*, the user engages in a more iterative strategy of trial-and-error. If the cost of a potential error (or computation) is high, spending more time in a preventative strategy may be optimal. In some cases, such as with streaming data or confidential data, a user may not have access to all of the data to which the transformation will be applied. In these cases, a preventative strategy may also be necessary.

Of course, it is often impractical for a user to address every possible type of failure condition for their transformation script and some balance is needed between applying preventative and iterative strategies. Future work might investigate algorithmic techniques for computing an optimal allocation of user attention given some constraints.

5.6.2 Coverage over the Wrangler DSL

Our work only applies to single record text manipulation transformations such as split, extract and cut. The Wrangler language contains other transforms that can also result in unintended results when applied to an entire data set. For instance, pivoting transformations create output columns from distinct values in input columns. If the sample does not

contain all distinct values present in the entire data set, the pivot will create an unexpected number of columns. Future work should investigate both transform-specific debugging methods for other transformations as well as generalized schemes that might be applied across transformations.

Debugging many-to-many transformations increases the complexity of identifying and presenting input records that led to an anomaly. For single record transformations such as split, a user can step back easily through history to identify the input record that led to the anomaly. However, with transformations such as pivot transformations, an anomalous record may be the result of applying a transformation to a large percentage of the input records. In these cases, efficiently tracking record provenance to identify contributing input records can be difficult [51]. Additionally, user interface design will need to help users assess this potentially large set of records through summarization or other techniques.

5.6.3 Multi-Step Transformation Scripts

Both debugging methods help an end-user debug a single data transformation at a time. Of course, most transform scripts consists of many dependent transformations that introduce additional considerations. Transformations that operate on index-based positions may have unexpected behavior when following other transformation steps. For instance, filtering the first row of a data set after a many-to-many transformation (e.g., sorting), may have unexpected consequences on a sample. Debugging multi-step transformations also introduces additional challenges for interface design. For instance, should a tool flag anomalies produced by one transformation that are edited and potentially fixed by downstream transformations?

5.6.4 Scaling Anomaly Assessment

In some cases there are too many possible exceptions to display to a user. Future iterations of the interface need to explore new methods for choosing representative subsets of exceptions and other summarization techniques. For instance, by employing techniques such as active learning [95], the system can choose an optimal set of samples to support certain tasks. For instance, to improve the accuracy of a classification model, the interface might

choose sample records near the decision boundary for the user to label. Additionally, clustering techniques might be used to group exceptions into exception classes, reducing the number of records a user needs to inspect. Profiler, discussed in the next chapter, addresses some of these limitations. In particular, Profiler uses summary visualizations to display sets or records that contain anomalies. Profiler's extensible framework permits the application of routines such as *Surprise-based anomaly detection*.

Chapter 6

Profiler: Integrated Analysis and Visualization

Data quality issues such as missing, erroneous, extreme and duplicate values undermine analysis and are time-consuming to find and fix. Automated methods can help identify anomalies, but determining what constitutes an error is context-dependent and so requires human judgment. While visualization tools can facilitate this process, analysts must often manually construct the necessary views, requiring significant expertise. We present Profiler, a visual analysis tool for assessing quality issues in tabular data. Profiler applies data mining methods to automatically flag problematic data and suggests coordinated summary visualizations for assessing the data in context. The system contributes novel methods for integrated statistical and visual analysis, automatic view suggestion, and scalable visual summaries that support real-time interaction with millions of data points. We present Profiler's architecture — including modular components for custom data types, anomaly detection routines and summary visualizations — and describe its application to motion picture, natural disaster and water quality data sets.

6.1 Introduction

Data sets regularly contain missing, extreme, duplicate or erroneous values that can undermine the results of analysis. These anomalies come from various sources, including

human data entry error, inconsistencies between integrated data sets, and sensor interference. Flawed analyses due to dirty data are estimated to cost billions of dollars each year [22]. Discovering and correcting data quality issues can also be costly: some estimate cleaning dirty data to account for 80 percent of the cost of data warehousing projects [19].

The statistics and database communities have contributed a number of automated routines for detecting dirty data, such as finding outliers or duplicate records. While these techniques can reveal potential issues, human judgment is required to determine if the issues are in fact errors and how they should be treated. For example, outlier detection might flag a high temperature reading; an analyst then needs to assess if the reading is an exceptional event or an error.

Discovering a potential error is only the first step towards clean data. Before manipulating the data, an analyst may investigate why an anomaly has occurred to inform possible fixes. The analyst must place the anomaly in context by scrutinizing its relationship with other dimensions of the data. Appropriately-chosen visualizations can help reveal and contextualize these anomalies. Histograms and scatter plots, for instance, may reveal outlying values in a distribution. Analysts typically have to choose which views to construct: they must determine which subset of data columns and rows to visualize, how to transform the data, choose visual encodings, and specify other criteria such as sorting and grouping. Determining which visualizations to construct may require significant domain knowledge and expertise with a visualization tool.

In response we present *Profiler*, a visual analysis system to aid discovery and assessment of data anomalies. Profiler uses type inference and data mining routines to identify potential data quality issues in tabular data. Profiler then suggests coordinated, multi-view visualizations to help an analyst assess anomalies and contextualize them within the larger data set.

Our first contribution is an **extensible system architecture** that enables integrated statistical and visual analysis for data quality assessment. This modular architecture supports plug-in APIs for data types, anomaly detection routines and summary visualizations. We populate this framework with commonly-needed data types and detection routines. We focus primarily on univariate anomalies due to their frequency, tractability, and relative ease of explanation. We demonstrate how coupling automated anomaly detection with linked

summary visualizations allows an analyst to discover and triage potential causes and consequences of anomalous data.

Our architecture also introduces novel visual analysis components. We contribute a technique for **automatic view suggestion** based on mutual information. Profiler analyzes the mutual information between table columns and the output of anomaly detection to suggest sets of coordinated summary visualizations. Our model recommends both table columns and aggregation functions to produce visual summaries that aid assessment of anomalies in context.

We also contribute the design of **scalable summary visualizations** that support brushing and linking to assess detected anomalies. Through linked selections, analysts can project anomalies in one column onto other dimensions. Our aggregate-based summary views bin values to ensure that the number of visual marks depends on the number of groups, not the number of data records. We provide optimizations for query execution and rendering to enable real-time interaction with data sets in excess of a million rows.

6.2 Usage Scenario

Before describing Profiler's architecture, we begin with a representative usage scenario. Consider an example task, using movie data compiled from IMDB, Rotten Tomatoes and The Numbers. This data set contains 16 columns and over 3,000 movies. The data includes metadata such as the title, primary production location, director, MPAA rating, and release date; financial information such as DVD sales and worldwide gross; and IMDB ratings.

An analyst is interested in which factors affect a movie's eventual revenue. She first loads the data into Profiler to assess overall data quality. The interface shows a schema browser, anomaly browser, formula editor and an empty canvas (Figure 6.1). The schema browser shows the column names in the data set; the analyst could double-click column names or drag them into the canvas to visualize the corresponding column. Instead, she examines the anomaly browser.

The anomaly browser displays potential quality issues, grouped by issue type and sorted by severity. For each issue, Profiler lists the columns containing the issue and the name of the detection routine that flagged the anomaly. The analyst clicks the MPAA Rating label

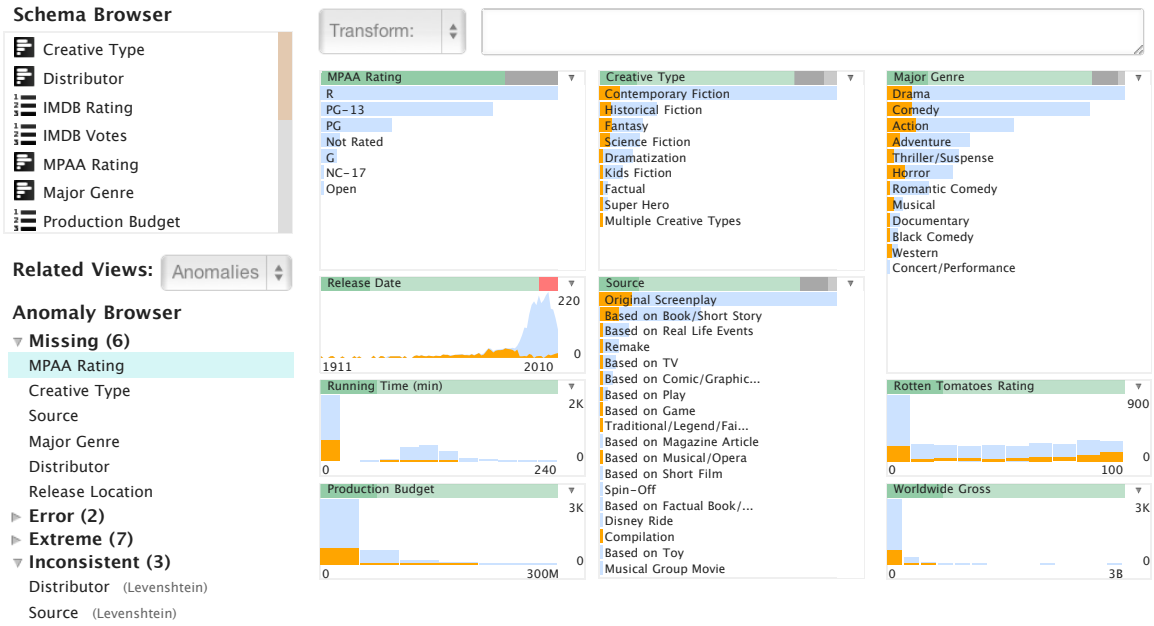


Figure 6.1: The Profiler User Interface. The UI contains (clockwise from top-left): (a) schema browser, (b) formula editor, (c) canvas of linked summary visualizations, and (d) anomaly browser. Profiler generates a set of linked views for each identified anomaly. Here, we investigate possible causes of missing MPAA movie ratings. The grey bar above the MPAA rating chart indicates missing values; we select it to highlight matching records. The Release Date chart shows that missing ratings correlate with earlier release dates.

in the missing values group. In response, Profiler displays the MPAA Rating data as a categorical bar chart showing the counts for each rating type. The chart title includes a data summary bar: green bars indicate parsed values, red bars indicate type verification errors, and grey bars indicate missing values.

Curious why so many values are missing, the analyst adds related visualizations by selecting the 'Anomaly' option in the related views menu — this operation requests views that might explain the observed anomaly. She then selects the grey bar in the MPAA Rating chart to see how missing values project across other columns (Figure 6.1). She finds that missing ratings correlate with early release dates. While this is interesting, she determines that the missing values don't have a strong relationship with any financial figures. This result holds for other columns with missing data.

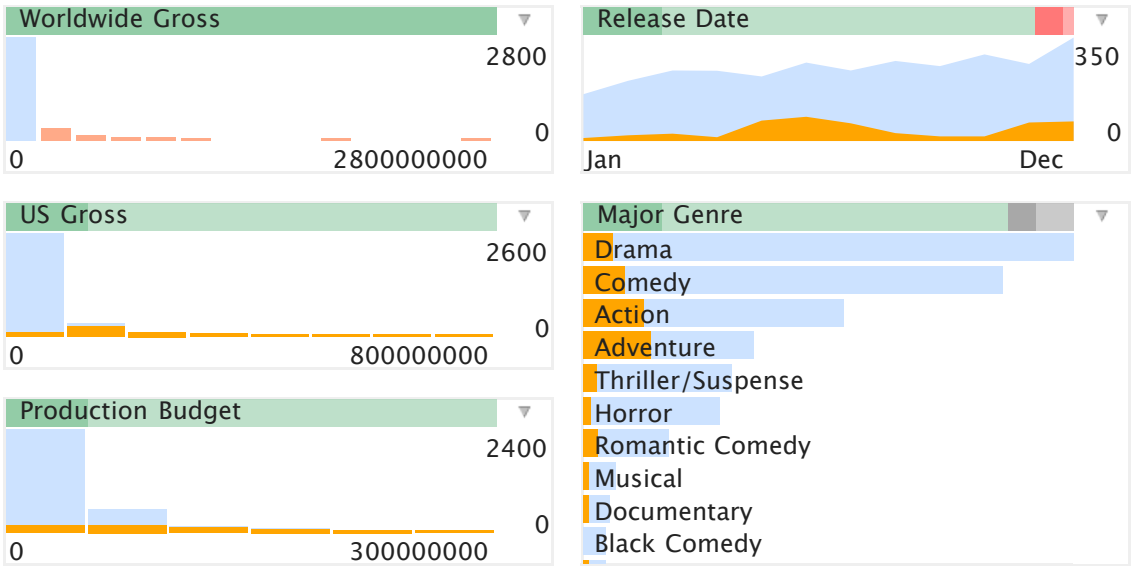


Figure 6.2: Automatically generated views to help assess Worldwide Gross. Worldwide Gross correlates with high US Gross and Production Budgets. High gross also coincides with Action & Adventure movies and the Summer & Winter seasons. Profiler chose to bin Release Date by month instead of by year.

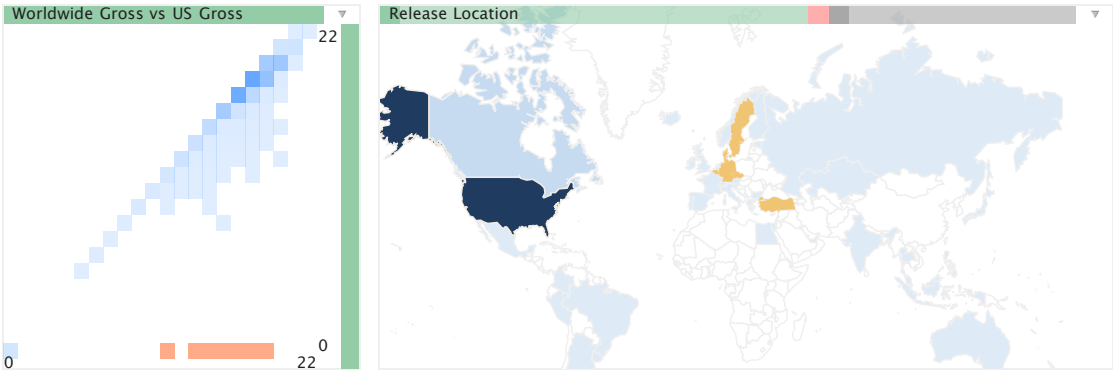


Figure 6.3: Map assessing 2D outliers in a binned scatter plot. Selected in the scatter plot are movies with high Worldwide Gross but low US Gross (in orange). Linked highlights on the map confirm that the movies were released outside of the US.

The analyst next decides to look at extreme values in financial figures and clicks Worldwide Gross in the 'Extreme' anomaly list. A histogram reveals a small number of high

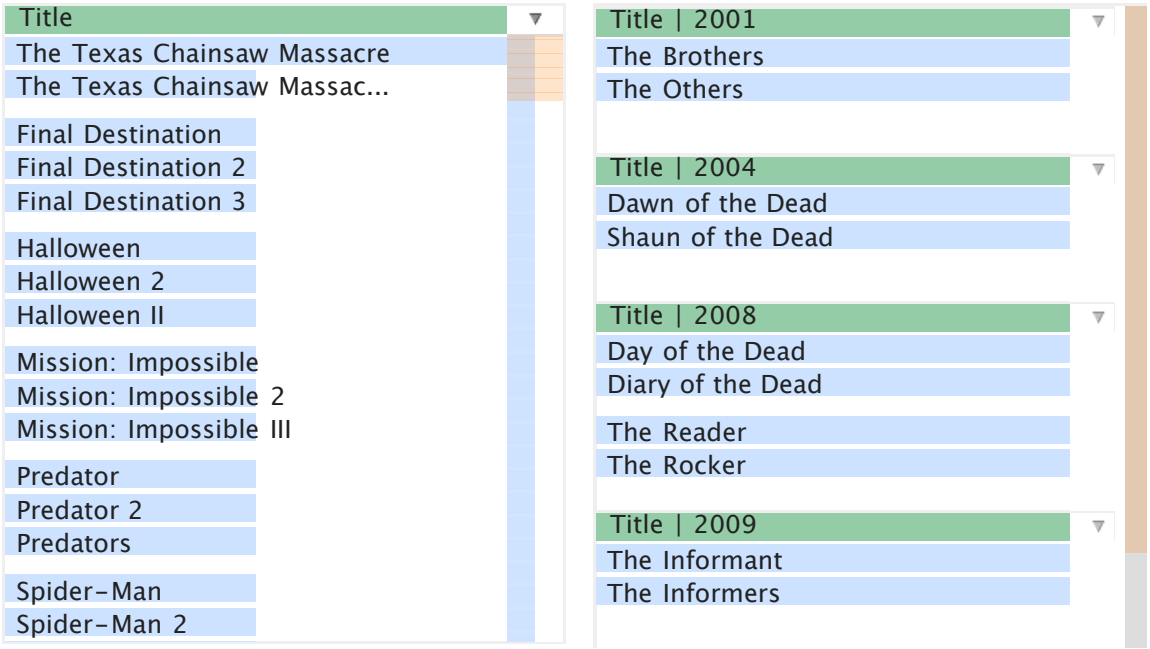


Figure 6.4: Conditioned duplicate detection. Left: Movie titles clustered by Levenshtein distance reveal over 200 potential duplicates. Right: Conditioning the clustering routine on 'Release Year' reduces the number of potential duplicates to 10.

grossing movies. To generate explanatory visualizations, the analyst selects 'Data Values' from the related views menu — this operation requests views that might help explain the total distribution of Worldwide Gross, not just flagged anomalies. She mouses over the bars at the high end of the Worldwide Gross histogram and sees that these values correlate with high values in other financial figures, such as U.S. Gross (Figure 6.2). She notices that Action and Adventure movies account for a disproportionate number of highly grossing movies. The time-series view reveals that these films spike during the summer and holiday seasons. The view groups release dates by month rather than year, as binning by month produces a stronger relationship with Worldwide Gross. The analyst is now confident that the outliers represent exceptional performance, not errors in the data.

The analyst decides to explore the seemingly strong relationship between Worldwide Gross and U.S. Gross. The analyst first selects 'None' in the related views menu to declutter the canvas. She drags U.S. Gross from the schema viewer onto the histogram displaying Worldwide Gross to create a binned scatterplot. The data appear to be log-normally

distributed so she uses the chart menu to set log scales for the axes. She notes outlying cells containing very low U.S Gross values compared to Worldwide Gross. She adds a map visualization by dragging Release Location to the canvas and confirms that most of these movies were released outside the U.S (Figure 6.3). The analyst decides to filter these data points from the data set so she chooses a *filter* transform from the transformation menu. The formula editor shows a predicate based on the current selection criteria and the analyst hits return to filter the points.

The analyst notices that the Release Location map contains a red bar indicating erroneous country values. She decides to toggle the map visualization to a bar chart to inspect the erroneous values. She clicks the small arrow at the top-right of the chart to open the chart menu and changes the visualization type. She filters the bar chart to only show erroneous values and sees a few 'None' and 'West Germany' values. To fix these errors, the analyst selects a *replace* transform in the formula editor menu and then specifies parameters; e.g., replace(Release Location, 'West Germany', 'Germany').

Next, the analyst inspects the 'Inconsistency' list in the anomaly browser. The analyst clicks on Title in order to spot potential duplicate records. Profiler responds by showing a grouped bar chart with movie titles clustered by textual similarity (Figure 6.4). Unsurprisingly, the analyst sees that movies and their sequels are clustered together. There also appear to be potential remakes of classic films. The analyst worries that there might also be misspellings of some films, but does not want to verify all the clusters by hand. The analyst reasons that true duplicates are likely to have the same Release Date and so decides to condition the text clustering anomaly detector on Release Date. The analyst clicks 'Levenshtein' next to Title in the anomaly browser. A menu appears which includes selection widgets for conditioning anomaly detection on another column. After rerunning the detector, there are significantly fewer anomalies to check. The analyst is satisfied that there are no duplicate entries and continues with her analysis.

6.3 System Architecture

Underlying the Profiler application is an extensible architecture that combines statistical algorithms and coordinated visualizations. The system is implemented in JavaScript, and

is intended to run inside browsers with optimized JavaScript execution engines. The architecture consists of five major components.

First, Profiler represents *data tables* using a memory-resident column-oriented relational database. The database supports standard SQL-style queries for filtering, aggregation, and generating derived columns. Unlike standard SQL databases, Profiler uses a relaxed type system: values can deviate from their column's defined type. Profiler flags these values as inconsistent; they appear in red within a chart's quality summary bar. The same database system also powers the Wrangler [57] data transformation tool. Profiler has access to the Wrangler data transformation language and extends it with additional transforms, including more advanced aggregation operations such as binning numeric data to compute histograms and mathematical operations for deriving new columns.

The rest of the Profiler architecture consists of four modular components (Figure 6.5). The *Type Registry* contains data type definitions and a type inference engine. Profiler uses types to choose appropriate anomaly detection routines and visualizations. The *Detector* performs anomaly detection by combining type-aware feature extractors and a set of data mining routines. Using detected anomalies and the mutual information between columns, the *Recommender* suggests visualizations to help an analyst assess potential issues. The *View Manager* presents linked summary visualizations; it generates type-specific visualizations and executes coordinated queries across views to support brushing and linking. We now describe each of these components in detail.

6.3.1 Type Registry

The *Type Registry* consists of a set of type definitions and routines for type inference. Each column in a data table is assigned a type, whether automatically via inference or manually by the user.

At minimum, a Profiler type is defined by a *binary verification function*: given an input value, the function returns true if the value is a member of the type and false otherwise. Verification functions include regular expression matches, set membership (e.g., dictionary lookup of country names) and range constraints (e.g., pH between 0-14). Profiler associates a type with an entire column, but not all values in the column necessarily satisfy the type

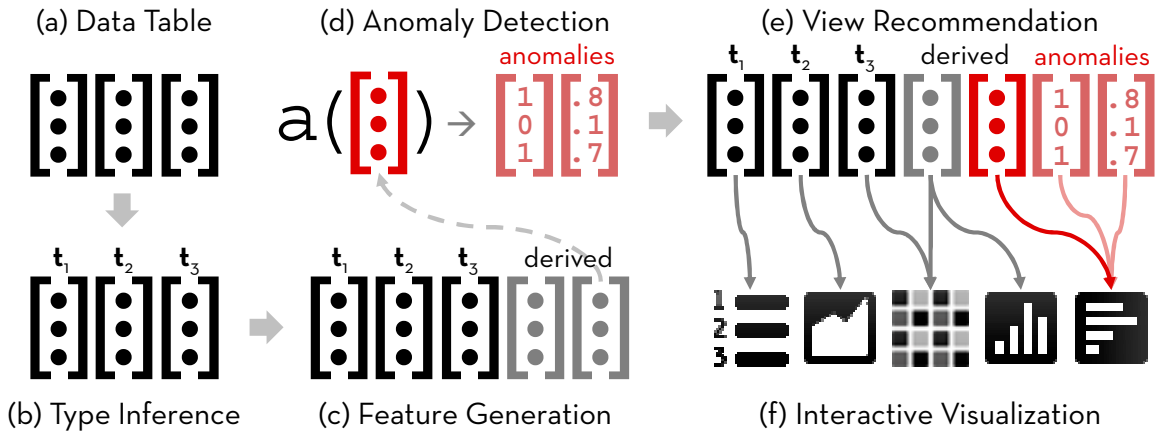


Figure 6.5: The Profiler Architecture. An (a) input table is analyzed to (b) infer types for each column. Type information is used to (c) generate features prior to running (d) anomaly detection routines. The results of anomaly detection and mutual information analysis are used to perform (e) view recommendation and populate a set of (f) interactive visualizations.

definition.

Profiler includes built-in support for primitive types — boolean, string, and numeric (int, double) — and higher-order types such as dates and geographic entities; e.g., state/country names, FIPS codes, zip codes. Profiler's detector and view manager components require that all columns be assigned to a data type. The type system is extensible: as new types are defined, anomaly detection and visualization methods can be specified in terms of pre-existing types or new components (e.g., a novel type-specific visualization) that plug-in to the Profiler architecture.

A type definition may also include a set of *type transforms* and *group-by functions*. A type transform is a function that maps between types (e.g., zip code to lat-lon coordinate). These functions form a graph of possible type conversions, some of which may be lossy. User-defined types can include type transforms to built-in types to leverage Profiler's existing infrastructure. Group-by functions determine how values can be grouped to drive scalable visualizations. For instance, numeric types can be binned at uniform intervals to form histograms, while dates may be aggregated into meaningful units such as days, weeks, months or years.

Type inference methods automatically assign a type to each column in a data table

Type	Issue	Detection Method(s)	Visualization
Missing	Missing record	Outlier Detection | Residuals then Moving Average w/ Hampel X84	Histogram, Area Chart
		Frequency Outlier Detection | Hampel X84	Histogram, Area Chart
	Missing value	Find NULL/empty values	Quality Bar
Inconsistent	Measurement units	Clustering | Euclidean Distance	Histogram, Scatter Plot
		Outlier Detection | z-score, Hampel X84	Histogram, Scatter Plot
	Misspelling	Clustering | Levenshtein Distance	Grouped Bar Chart
	Ordering	Clustering | Atomic Strings	Grouped Bar Chart
	Representation	Clustering | Structure Extraction	Grouped Bar Chart
	Special characters	Clustering | Structure Extraction	Grouped Bar Chart
Incorrect	Erroneous entry	Outlier Detection | z-score, Hampel X84	Histogram
	Extraneous data	Type Verification Function	Quality Bar
	Misfielded	Type Verification Function	Quality Bar
	Wrong physical data type	Type Verification Function	Quality Bar
Extreme	Numeric outliers	Outlier Detection | z-score, Hampel X84, Mahalanobis distance	Histogram, Scatter Plot
	Time-series outliers	Outlier Detection | Residuals vs. Moving Average then Hampel X84	Area Chart
Schema	Primary key violation	Frequency Outlier Detection | Unique Value Ratio	Bar Chart

Figure 6.6: Taxonomy of Data Quality Issues. We list classes of methods for detecting each issue, example routines used in Profiler, and visualizations for assessing their output.

based on the Minimum Description Length principle (MDL) [87]. MDL selects the type that minimizes the number of bits needed to encode the values in a column. MDL has been used effectively in prior data cleaning systems, such as Potter's Wheel [86]. We use the same MDL formulation in Profiler.

6.3.2 Detector

Profiler's *Detector* applies a collection of type-specific data mining routines to identify anomalies in data.

The Detection Pipeline

The Detector determines which anomaly detection routines to apply, runs them, and produces output for visualization. This process has two phases: feature generation and anomaly detection.

During feature generation, the Detector derives *features* of the input columns to use as input to anomaly detection routines. Features are extracted using unary transformations called *generators*. For example, a generator might compute the lengths of string values; an anomaly detector might then compute z-scores to flag abnormally long strings. The Detector maintains a list of appropriate generators (including the identity function) for each type in the Type Registry. Given an input table, the Detector applies generators to each input column according to its type signature. The result is a set of feature columns that serve as input to anomaly detectors.

Detection routines then analyze the feature columns. Detection routines accept columns as input and output two columns: a *class* column and a *certainty* column. The class column contains integers; 0 indicates that no anomaly was found in that row. Non-zero values indicate the presence of an anomaly and distinct integers indicate distinct classes of anomaly. For example, the z-score routine outputs a class column where each value is either 0 (within 2 standard deviations from the mean), -1 (< 2 stdev), or 1 (> 2 stdev). The certainty column represents the strength of the routine's prediction. For z-scores, these values indicate the distance from the mean.

The Detector organizes detection routines by the data types they can process. After feature generation, the system visits each column in the data table (including derived columns) and runs all routines with a compatible type. For instance, the z-score routine is applied to all numeric columns. The standardized output of class and certainty columns is then handled in a general fashion by the downstream Recommender and View Manager components.

The Detector's output appears in the anomaly browser. This browser lists any result of a detection routine that contains at least one anomalous value (i.e., a non-zero value in the class column), grouped by the type of detection routine and sorted by decreasing anomaly

count. The browser displays the columns containing the anomaly and which routines detected the anomaly. When a user clicks an item, relevant views appear in the canvas.

Detection Routines

Profiler incorporates a variety of detection routines to flag data anomalies (Figure 6.6), and can easily be extended with new routines. The following list focuses on the most common needs and demonstrates the diversity of routines that the system supports.

Missing value detection identifies cells that do not contain data.

Type verification functions identify values inconsistent with a given column type (Section 6.3.1). Verification can flag incorrect use of physical types (e.g., strings vs. integers) or constraint violations.

Clustering is used to detect a variety of errors relative to a chosen distance metric. Euclidean distance is useful for detecting numeric outliers and inconsistent measurement units. Character-based (Levenshtein distance), token-based (Atomic Strings), and phonetic-based (soundex) distances are useful for detecting inconsistencies in text such as misspellings, different term orderings, and phonetically similar words [23]. We use nearest neighbor agglomerative hierarchical clustering with each distance metric.

Univariate outlier detection routines identify extreme and possibly incorrect values for numeric and time-based data. We apply both z-scores and Hampel X84 — a routine based on median absolute deviation — to detect univariate quantitative outliers [41].

Frequency outlier detection identifies values that appear in a set more or less often then expected. Frequency outliers are commonly used to detect primary key violations. Profiler uses the unique value ratio to detect primary keys [41]. We use numerical outlier routines on aggregated counts to detect other types of anomalies, such as gaps in ranges which may indicate missing observations.

Profiler supports two methods of multivariate outlier detection. First, detection routines can accept multiple columns as input. For example, Mahalanobis distance can be used to detect multivariate numeric outliers [41]. Second, *conditioning* is a general method for converting any routine into a multivariate routine. Conditioning applies an existing routine to subsets of data, grouped by one or more variables (e.g., categorical or binned quantitative values). For instance, conditioning the z-score routine on genre calculates the

scores for values within each genre separately. To support conditioning, Profiler uses a *partitioner* that applies any transformation to data subsets formed by applying specified group-by functions.

The space of possible routines is combinatorially large and the results of these routines can be difficult to interpret. As a result, Profiler does not automatically run multivariate outlier detection routines by default. Users can initiate multivariate outlier detection by adding conditioning columns to existing univariate detectors.

6.3.3 View Recommendation

For a given anomaly, the *Recommender* automatically populates the View Manager (discussed next) with relevant visual summaries. Generating summary views requires recommending a *view specification* — a set of columns to visualize and type-appropriate group-by functions for aggregation. A view specification can also include anomaly *class* and *certainty* columns to parameterize a view. The recommender always specifies a *primary view* that visualizes the column(s) that contain the anomaly. The recommender also determines a set of related views using a model based on mutual information. The Recommender supports two types of related views. *Anomaly-oriented views* show columns that predict the presence of anomalies. *Value-oriented views* show columns that best explain the overall distribution of values in the primary column(s). Users select which type of view to show with the related view menu.

Mutual Information

The mutual information of two variables quantifies how much knowing the value of one variable reduces the uncertainty in predicting a second variable. It is equivalent to the reduction in entropy attained by knowing a second variable. Mutual information is non-negative and has a minimum of 0 when variables are independent. To compare mutual information across pairs of variables, we define a distance metric D that is 0 for completely dependent variables and increases as the mutual information between variables decreases. For variables X and Y with mutual information $I(X,Y)$ and entropies $H(X)$ and $H(Y)$, we define D as:

$$D(X,Y) = 1 - \left(\frac{I(X,Y)}{max(H(X),H(Y))} \right) \tag{6.1}$$

Recommendation

We use the metric D to recommend both the primary view and related views. A view specification determines how data is aggregated for a visual summary by assigning each row of input a group id (e.g., a bin in a histogram or binned scatterplot). In this way, we can derive a column of group ids from a view specification. We define $ViewToColumn$ as a function that converts a view specification into a column of group ids. For a set of columns C, we use VS_C to refer to the set of all possible view specifications containing one column from C and a type-appropriate group-by function.

The primary view always displays the set of columns that contain the anomaly. Our goal is to produce a summary view with bins that minimize the overlap of anomalies and non-anomalies so that analysts can better discriminate them. Recall that the *class* column output by the Detector indicates the presence of anomalies. We enumerate pairs of $\{column, group\text{-}by\ functions\}$ and select the pair that best predicts the *class* column. More formally, if A is the set of columns containing the anomaly, we recommend the view specification $vs \in VS_A$ that minimizes the quantity $D(ViewToColumn(vs), class)$. This primary view specification (denoted pvs) is assigned the *class* and *certainty* columns as parameters.

To suggest *anomaly-oriented views*, we find other columns that best predict the *class* column. We consider the set of all columns R that exclude the columns in C. We then choose view specifications from VS_R that predict the *class* column. We sort specifications $vs \in VS_R$ by increasing values of $D(ViewToColumn(vs), class)$. The Recommender populates the View Manager with the corresponding visual summaries in sort order until the canvas is full, discarding summaries that contain columns already visualized.

We use a similar process to recommend *value-oriented views*. Value-oriented views show visualizations related to the entire distribution of values in the primary view, not just anomalies. Instead of predicting the *class* column, we predict the group ids generated by the primary view specification. We sort specifications $vs \in VS_R$ by $D(ViewToColumn(vs), ViewToColumn(pvs))$. Because VS_R only contains view specifications with one column, only univariate summaries are suggested. Our approach extends to multiple columns if we

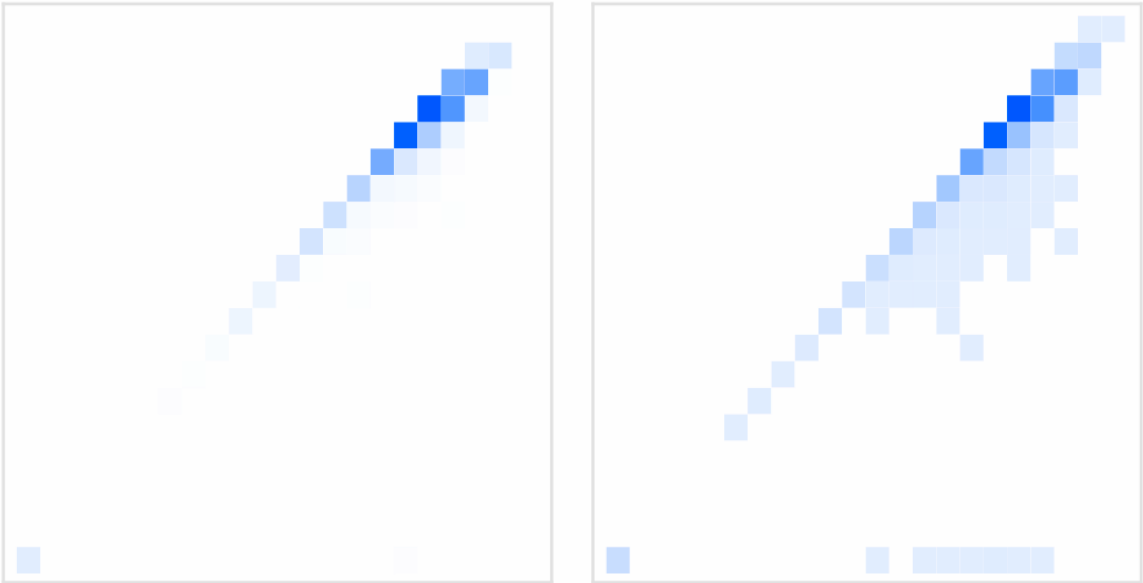


Figure 6.7: Adding perceptual discontinuity to summary views. Left: A binned scatter plot using a naive opacity ramp from 0-1. Right: An opacity scale with a minimum non-zero opacity ensures perception of bins containing relatively few data points.

augment R to include larger subsets of columns.

6.3.4 View Manager

The *View Manager* converts view specifications into a set of linked visual summaries. The View Manager creates type-specific views to reveal patterns such as gaps, clusters and outliers. A query engine for filtering and aggregating data supports rapid brushing and linking across summaries, allowing an analyst to determine how subsets of data project across other dimensions. In addition to automatic view recommendation, analysts can manually construct views through drag-and-drop and menu-based interactions. Profiler visualizations are implemented in SVG using the D3 library [7]. We now detail the design of the View Manager, including optimizations for rendering and query performance.

Summary Visualizations

Visualizing "raw" data is increasingly difficult with even moderately sized data — even a few hundred data points may make a scatter plot difficult to read due to overplotting. Profiler's summary visualizations use aggregation to scale to a large number of records [10, 49, 64, 89, 105]: the number of marks in each view depends primarily on the number of bins, not the number of records.

To compute aggregates, each view requires a group-by function that specifies a binning strategy. For automatically generated views, bins are determined by the Recommender. When a user manually selects columns to visualize, Profiler chooses a group-by function based on the range of data values. Users can also select group-by functions or type transformations through a view's context menu.

Histograms (numeric data), **area charts** (temporal data), **choropleth maps** (geographic data) and **binned scatter plots** (2D numeric or temporal data) visualize counts of values within binned ranges. Though Carr [10] recommends hexagonal binning of scatter plots for improved density estimation, we currently use rectangular binning to enable better query and rendering performance.

Profiler uses **bar charts** to visualize the frequencies of distinct nominal values. Sorting the bars by frequency helps analysts assess the distribution of values within a set. **Grouped bar charts** display the frequencies of clustered values (e.g., clusters of possible duplicate values). For columns with high cardinality, it is not feasible to show all the bars at once. In response, Profiler also visualizes the distribution in the chart's scroll bar. We perform windowed aggregation over contiguous bars to form summary counts; the window size is adjusted as needed to match the available pixel resolution.

Data quality bars summarize column values as valid, type errors, or missing. Profiler annotates each visualization with one or more quality bars to indicate missing or non-conforming data. Quality bars also act as input regions for brushing and linking.

Higher-dimensional views are depicted using **small multiples**. Any Profiler visualization can be used in a trellis plot, with subplots showing data subdivided by one or more conditioning variables. Finally, Profiler's **table display** presents the raw data. Analysts can filter table rows by brushing summary visualizations.

Profiler visualizations also incorporate design considerations for varying levels of scale. Naïve scaling of bar heights and color ramps can result in low-frequency bins that are essentially invisible due to minuscule bars or near-white colors. This is unacceptable, as outliers often reside in low-frequency bins. We induce a perceptual discontinuity in these scales so that low-frequency bins remain identifiable: we give small bars a minimum height and make colors for any non-zero values suitably distinguishable from the background (Figure 6.7). In addition, different tasks may require visualizing data at different levels of detail. Profiler time-series charts support binning by a variety of time spans (day, month, year, etc). Maps include panning and zooming controls.

Each view can be parameterized using the *class* and *certainty* columns generated by an anomaly detector. The bar chart and small multiples views enable sorting by *class* and *certainty*. By default we sort in descending order to reveal anomalies with higher certainty; e.g., a grouped bar chart will sort clusters of similar words by the *certainty* that the cluster contains misspelled items, with groupings determined by the *class* column.

Scalable Linked Highlighting

When a user selects a range of values (e.g., by mouse hover), Profiler highlights the projection of that data across all views. To do so, Profiler first filters the backing data table to include only the selected range. For each view Profiler then computes an aggregate query to determine the count of selected items in each bin. These data points are visualized as orange highlights overlaid over the original view (see Figure 6.1). Linked selection extends to all visualizations, including quality bars, scrollbars, and table views.

To support scalable, real-time interaction we addressed two performance challenges: query execution and rendering. To reduce the query load we first simplify the data. Multiple records in the input data often map to the same bin. In response we pre-aggregate the data, grouping by the bins for all visualized attributes. With a suitable number of bins per dimension (typically 10-30) this step can reduce the number of records by one to two orders of magnitude.

To further reduce query time, we encode non-numeric types as zero-based integers. Integer codes speed element comparisons and simplify the calculation of dimensional indices during aggregation. The original values are encoded in sort order and stored in a lookup

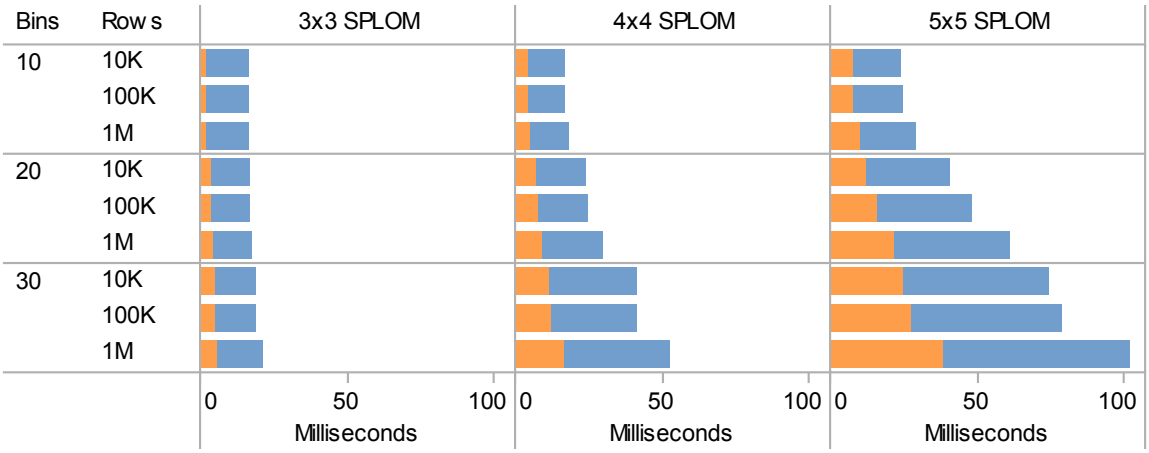


Figure 6.8: Performance (in ms) of linked highlighting in a scatter plot matrix (SPLOM). Orange bars represent query processing time, blue bars represent rendering time. We varied the number of dimensions, bins per dimension and data set size. In most cases we achieve interactive (sub-100ms) response rates.

table for reference. To facilitate optimization by the browser's just-in-time compiler, the inner loop of the query executor avoids function calls. We also cache query results to eliminate redundant computation. For example, in a scatter plot matrix (SPLOM) cross-diagonal plots visualize the same data, only transposed.

Rendering bottlenecks also limit performance. Even with aggregated views, the number of marks on-screen can grow large: a 4×4 SPLOM containing plots with 50×50 bins requires rendering up to 40,000 marks. To speed rendering we minimize modifications to the Document Object Model (DOM) in each interactive update. To avoid churn, we introduce all SVG DOM elements (including highlights) upon initialization. Each update then toggles a minimal set of visibility and style properties. We also try to take advantage of optimized rendering pathways, for example by using squares instead of hexagons in binned scatter plots.

Performance Benchmarks

We benchmarked query and render times during interactive brushing and linking. For our test data, we sample from random distributions for up to five columns. Three of the columns are independently normally distributed. The others are linearly or log-linearly dependent

with the first column. We visualize the data as a SPLOM with univariate histograms along the diagonal. We then programmatically brush the bins in each univariate histogram. This approach provides a conservative estimate of performance, as brushing scatter plot bins results in smaller selections and hence faster processing. We varied the number of visualized columns (3, 4, 5), bin count (10, 20, 30), and data set size (10K, 100K, 1M rows). For each condition, we averaged the query and render times across 5 runs. The benchmarks were performed in Google Chrome v.16.0.912.75 on a quad-core 2.66 GHz MacPro (OS X 10.6.8) with per-core 256K L2 caches, a shared 8MB L3 cache and 8GB RAM.

Our results demonstrate interactive rates with million-element data sets (Figure 6.8). We see that the number of columns and number of bins have a greater impact on performance than the number of data points. We also performed an analysis of variance (ANOVA) to assess the contribution of each factor to the average response time. We found significant effects for SPLOM dimension ($F_{2,20} = 21.4$, $p < 0.0001$) and bin count ($F_{2,20} = 14.8$, $p = 0.0001$). However, we failed to find a significant effect of data set size ($F_{2,20} = 1.2$, $p = 0.3114$), lending credence to our claim of scalability.

6.4 Initial Usage

We have conducted informal evaluations of Profiler on a variety of data sets — including water quality data, a disasters database, obesity data, a world wide quality-of-life index, and public government data. We now describe two concrete examples of how Profiler has enabled rapid assessment of these data sets.

The disasters database contains 11 columns, including the type, location, and consequences (cost, number affected) of the disaster. Profiler identified 13 data quality issues. These include 2 columns containing duplicates due to inconsistent capitalization, 6 columns with missing values, and 3 columns with extreme values. For example, Profiler detected disasters with extremely high monetary cost. The recommended views include the Type column. Upon selecting large values in the Cost histogram, it became evident that the vast majority of these outliers were floods, storms or droughts. By selecting these values in the Type bar chart, we confirmed that these disaster types typically lead to higher cost. For columns with missing values, Profiler primarily recommends columns with co-occurrences

of missing values. For instance, rows missing values in a Killed column also tend to have missing values in the Cost, Sub Type, and Affected columns. Because of this, the recommended views for each of these anomalies were very similar. Assessing data quality in this data set took just a few minutes.

We also tested Profiler on World Water Monitoring Day data. Each year, thousands of people collect water quality data using test kits; they manually record the name and location of the body of water as well as measurements such as Turbidity and pH. The data contains 34 columns. Profiler identifies 23 columns with missing data, 2 with erroneous values, 5 containing outliers and 5 containing duplicates. For instance, the Air Temperature column contains extremely low temperatures. Profiler recommends a world map and a visualization of the date of collection, revealing that the extreme lows were collected in Russia during winter. The data set also contains many duplicates. Data collectors often refer to the same city by slightly different names, resulting in hundreds of potential duplicates. After inspecting a few duplicate sets, we conditioned text clustering on the State column to simplify the clustered bar charts significantly. However, Profiler also flagged possible duplicates in the State column itself, prompting us to resolve duplicates there first. Profiler also flagged the Site Country name for containing erroneous country names; a recommended bar chart shows that people enter extra specifics, such as "Congo, Republic of (Brazaaville)." We then corrected these values to enable proper aggregation.

6.5 Conclusion

In this chapter we presented Profiler, an extensible system for data quality assessment. Our system architecture can support a flexible set of data types, anomaly detection routines and summary visualizations. Our view recommendation model facilitates assessment of data mining routines by suggesting relevant visual data summaries according to the mutual information between data columns and detected anomalies. We demonstrated how the appropriate selection of linked summary views aids evaluation of potential anomalies and their causes. We also discussed optimizations for scaling query and rendering performance to provide interactive response times for million element data sets within modern web browsers. By integrating statistical and visual analysis, we have found that Profiler can

reduce the time spent diagnosing data quality issues, allowing domain experts to discover issues and spend more time performing meaningful analysis.

Chapter 7

Conclusion

In this thesis, we identified processes data analysts frequently encounter that are underserved by visual analytic tools. In particular, despite the fact that analysts spend much of their time transforming data and inspecting it for data quality issues, most visual analytic tools assume data is well-formatted and do not support interactive transformation. At the same time, many analysts lack the programming experience necessary to manipulate large data sets. These analysts either rely on others in their organizations to complete their work or can not complete analysis at scale. In response, this dissertation contributes new systems and algorithms to enable analysts to wrangle data at scale.

7.1 Thesis Contributions

This thesis focused primarily on the problem of designing interactive systems to support the iterative process of data assessment and transformation. We synthesized results from an interview study with 35 professional data analysts (Chapter 2). We characterized the analysis process into 5 stages: discovery, wrangling, profiling, modeling and reporting. We identified recurring pain points within this process, finding that current visual analytic tools lack support for the early stages of the pipeline. Based on these finding, we developed tools to support iterative data transformation and data quality assessment.

We presented Wrangler (Chapter 4), a graphical system for data transformation. The

system contributes a novel method for suggesting transformations in an underlying, declarative transformation language from user interactions and mechanisms for rapidly evaluating these suggestions. We also conducted a controlled user study comparing Wrangler and Excel across a set of data wrangling tasks. We find that Wrangler significantly reduces specification time and promotes the use of robust transforms rather than manual editing.

Our work with Wrangler also identified important problems that led to follow-up work. First, users have difficulty specifying reshaping transformations. Second, users have difficulty formulating multi-step transformation strategies. Third, users who rely solely on visual previews can often not disambiguate transform suggestions on a given sample. Finally, it is difficulty to assess whether a transformation script, built on a sample, produces the desired results on a large data set. We addressed the first two problems with Proactive Wrangler [35].

We addressed the last two issues in our work on Debugging Transformation Scripts (Chapter 5). We contributed methods to aid users debugging transformation scripts. *Rule-based transform disambiguation* provided a mechanism for generating examples to help analysts refine transformations. We evaluated the complexity of this technique and discussed issues applying it. In the worst case, an end-user will need to evaluate $|T|-1$ synthetic example records to disambiguate a set of transforms T. User interfaces leveraging synthetic examples must make the distinction between syntactic and semantic correctness clear to end-users. *Surprise-based anomaly detection* leveraged a random forest classifier to classify output records as exceptions. We evaluated this model on a corpus of log files, and compared it to two other models. The random forest model had higher precision than the other models with comparable recall. All models performed worst on a set of expert chosen transformations, indicating the technique might be more applicable for novice users than expert users. We then discussed relative advantage of the two techniques. *Rule-based transform disambiguation* can provide synthetic examples without the cost applying transformations to any of the data. *Surprise-based anomaly detection* identifies potentially erroneous records, but requires applying a transformation to each of those records.

We also contributed Profiler (Chapter 6), a visual analytic system for assessing data

quality issues such as missing, erroneous, extreme and duplicate values. The system contributes novel methods for integrated statistical and visual analysis, automatic view suggestion, and scalable visual summaries that support real-time interaction with millions of data points. We presented Profiler's architecture, including modular components for custom data types, anomaly detection routines and summary visualizations.

7.2 Research Opportunities

Although this dissertation addresses critical challenges in the analytic pipeline, there are still many challenges that should be addressed by future work. Our interviews suggest analysts still struggle at many stages in their workflow. This thesis only addresses a subset of these stages. Below we discuss opportunities for future research spanning algorithms, visual analytics and social computation. We also identify areas that would benefit from better integration of automated routines with interactive interfaces.

7.2.1 Discovery

As the availability and diversity of data sources increases, so has the difficulty of finding and understanding data relevant to an analysis task. Large organizations may have hundreds or thousands of databases, which often lack documentation. Going forward, analytics tools should support visualizing and exploring these large, diverse data sets.

In these cases, analytic tools might consider entire tables or collections of tables as data points. For instance, a tool might create a summary of a table based on features of its columns. Of course, this challenge becomes more difficult when the schema of a data source is unknown.

Similarly, discovery tools should support exploring possible relationships between multiple data sources. For instance, can new systems help analysts discover data for augmenting an existing analysis? Past research has produced techniques for identifying relationships such as foreign-key relations between tables, but little work has explored how best to leverage these techniques in analytic tools.

7.2.2 Wrangling

Although Wrangler address some of the challenges in the wrangling phases, many challenges still remain. For instance, Wrangler only works over a single data source. Perhaps the most common wrangling challenge is integrating data from different sources. Integration requires being able to join or link data sets together along one or more shared dimensions. A number of algorithmic techniques have been developed to help the integration process: resolved entities or semantic data types may be used to match data together. A common subproblem is schema matching: mapping the dimensions of one data set onto the dimensions of another. However, even with matching data types, integration may be difficult. For example, how should one join sensor measurements taken at different time intervals? Future research might further investigate how visual interfaces and automated approaches to data integration could be more deeply combined.

Wrangler also lacks support for free text data and non-textual data, such as images and video. Future work should categorize the challenges working with these data sources and begin addressing issues such as exploring and extracting features from large mutlimedia collections.

7.2.3 Profiling

Our work on profiling leaves a number of opportunities for future work. For instance, Profiler supports a number of predefined data types. Future tools should enable end users to define custom types (c.f., [92]) and to incorporate detectors and visualizations for additional data types such as free-form text. Profiler's query engine is currently limited to data that fits within a browser's memory limit. Future work might examine hybrid approaches that combine server-side aggregation with client-side interactive querying. Our model for view recommendation currently uses pairwise mutual information, which is insensitive to redundant dependencies between data. Other methods, such as structure learning of Bayesian networks, might account for conditional dependencies between sets of columns to sidestep redundancy and further improve view ranking.

This thesis has argued that interactive visualizations can be a powerful tool for identifying and transforming data quality issues. However, once found, it is not always clear

if and how one should modify the data in response. In fact, some may wish to proceed with visual analysis despite the presence of missing data, outliers, or other inconsistencies. Such actions naturally raise the question: how can visualizations be best designed to support reasoning with dirty or uncertain data? As in data diagnosis, one would like errors such as missing data to be visibly indicated. However, unlike data diagnosis, one may wish to reduce their visual saliency so as not to unduly distract from analysis of the rest of the data. Though previous work has proposed methods for visualizing missing or uncertain data, little research has studied the effectiveness of these techniques.

For instance, the field would benefit from a deeper understanding of how various representations of uncertainty affect perception and reasoning. Consider the representation of uncertainty by blurring: Kosara [63] has found that people have difficultly identifying different levels of blur, implying that blur is a relatively ineffective encoding for multiple levels of uncertainty. However, most of the proposed solutions for visualizing uncertainty have not been empirically evaluated. Moreover, many techniques for handling uncertainty require choosing an underlying statistical model. Interactive visualization might aid in both selecting and evaluating such choices.

The goal of living with dirty data suggests important criteria for visual analysis systems. Do the data models provided by our systems explicitly support missing values or values which deviate from a schema? For example, a collection of numbers with a few erroneous string values interspersed should not prevent a tool from visualizing most values along a numeric axis. In such cases, the visualization might also include an indication of the presence and amount of deviant data. More advanced systems might also consider the semantics of uncertainty when transforming data—for example, how uncertainty propagates across aggregations and other analysis routines [5, 16]—and use this information to incorporate uncertainty into the visualization.

While our discussion has focused primarily on interactive tools, statisticians and database researchers have developed a number of analytic techniques for assessing data quality. These techniques include algorithms for detecting outliers and discrepancies [41, 45]. Other approaches range from simple validation routines (e.g., regular expression patterns) to complex data mining algorithms. How might we use visualization to best communicate the results of these routines? How can visual interfaces be used to specify or steer appropriate

routines based on the semantics of the data? Can visualizations also serve as an input device for authoring new validation patterns? Moreover, we might evolve these algorithms—e.g., using active learning approaches [91]—so that they can improve in response to guidance and verification from analysts.

7.2.4 Modeling

One clear implication of our studies is the need for visual analytic tools that scale to large, complex data sets. Producing such tools requires addressing a number of computational limitations. Future tools should leverage existing systems for large-scale interactive data processing whenever possible. At certain scales, interactively computing query results is not possible. Future work should leverage existing techniques for data sampling, approximate query processing, and pre-aggregation to support interactive response times. It remains future work to examine how low-latency query processing over data subsets of various resolutions impact both the quantity and quality of analysis.

With "wide" data sets, containing many data fields, it is extremely time-consuming to understand which fields are important for a given analysis. Going forward, visual analytic tools should support techniques for feature selection and deriving new features from input data. Such tools would require supporting both automated methods for feature selection, and enabling rapid evaluation of such methods.

7.2.5 Reporting

Analysts often have difficulty communicating all of the assumptions and transformations they made throughout an analysis. Analysts who use multiple tools also find it challenging to reproduce their analysis for others as well as perform "what if?" or sensitivity analyses. To address these challenges, future tools should augment intermediate products such as scripts and data with additional metadata. Such metadata might include the script's author, the rationale for an analysis procedure or assumptions made about the input data. For instance, a precise description of the changes made—and the rationales behind them—allows us to reconstruct the data wrangling process post-hoc and assess the impact of each change on the analysis. The metadata can enable more efficient search over products and

simplify the interpretation of results by others.

How to best capture, represent and interact with this metadata is an interesting visual analytics problem. Future work should assess how visual analytics tools might be designed to capture and exploit metadata. Should analysis tools only take data sets as input (as is typically done), or be extended to become "provenance-aware"? What is the right separation of concerns for tool modularity? System design questions arise both for lower-level performance issues—how to support rapid editing and rollback, e.g., by caching intermediate transformation states—and for user interface design—how might data transformations and annotations be surfaced in analysis tools to aid reasoning?

7.2.6 Leveraging Social Interaction

One of the insights motivating our interest in data wrangling tools is that algorithms are not enough—nuanced human judgements are often necessary throughout the process, requiring the design of interactive tools. One avenue for further reducing the costs associated with data preparation is to consider collaboration. To amortize wrangling costs and improve the scalability of data cleaning in the wild, we might cast data wrangling as an exercise in social computing.

As a first step, we can consider how the wrangling efforts of one analyst might be picked up and used by others. Indexing and sharing of data transformation scripts would allow analysts to reuse previous data wrangling operations, with the goals of saving time and improving data consistency. Transformation revisions submitted by other collaborators could improve the quality or reliability of shared transforms. By deploying wrangling tools on the public web, a large audience (analysts, journalists, activists, and others) might share their transformations, and thereby further open data access. Research challenges arise in how to search for, present, and suggest transformations, or transformation subsets, developed by others.

While the sharing of individual scripts has a clear utility, additional benefits might arise from analyzing a large corpus of wrangling scripts. For example, one could analyze data set features (e.g., data types, columns names, distributions of values) to learn mappings to likely transformations or infer higher-level semantic data types. This data could lead

to better automatic suggestions [8]. Such a corpus would also be a valuable resource for studying data cleaning strategies and informing the iterative design of wrangling tools.

Another opportunity lies in providing mechanisms for user-contributed type definitions: how can we best enable data domain experts to define new semantic data types? Analysts might author and share domain-specific data type definitions enabling verification, reformatting, and transformation (e.g., mapping between zip codes and latitude-longitude pairs). Incorporating domain-specific knowledge can improve validation and might also facilitate data integration. Though type authoring is likely feasible for only a cadre of advanced users, a broad class of analysts might benefit by applying those types to their data. We might look for guidance from existing systems for end-user authoring of data reformatting and validation rules [92].

Finally, we can consider how data quality might be improved by social interactions occurring across different phases of the data lifecycle. While data wrangling typically seeks to improve data quality prior to more sustained analyses, inevitably the process will be imperfect. Downstream analysts or visualization users, who might not have been involved in the initial data preparation, may also discover data errors. Indeed, such discoveries appear to be a common occurrence in social data analysis environments [40, 106]. What interaction techniques might allow such users to annotate, and potentially correct, data quality issues discovered during subsequent analysis? How can these discoveries be fruitfully propagated into data transformation scripts and brought to the attention of other users of the data?

7.3 Conclusion

Looking forward, this thesis addresses only a subset of the hurdles faced by data analysts. As data processing has become more sophisticated, there has been little progress on improving the tedious parts of the pipeline: data discovery, data entry, data (re)formatting, data cleaning, etc. The result is that people with highly specialized skills (e.g., statistics, molecular biology, micro-economics) spend more time in tedious data cleaning tasks than they do in exercising their specialty, while less technical audiences are unnecessarily shut out. We believe that more research integrating methods from HCI, visualization, databases, and statistics can play a vital role in making data more accessible and informative.

Bibliography

[1] McKinsey Quarterly, 2009.

[2] P. D. Allison. *Missing Data*. Sage Publications, 2001.

[3] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *Proc. IEEE Information Visualization (InfoVis)*, pages 111–117, 2005.

[4] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD*, pages 337–348, 2003.

[5] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs: databases with uncertainty and lineage. In *VLDB '06: 32nd international conference on Very large data bases*, pages 953–964. VLDB Endowment, 2006.

[6] Alan F. Blackwell. SWYN: A visual representation for regular expressions. In *Your Wish is my Command: Programming by Example*, pages 245–270, 2001.

[7] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE TVCG*, 17(12):2301–2309, 2011.

[8] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. VisTrails: visualization meets data management. In *Proc. ACM SIGMOD*, pages 745–747, 2006.

[9] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large N. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[10] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large N. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):1–58, July 2009.

[12] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 313–324, New York, NY, USA, 2003. ACM.

[13] George Chin, Olga A. Kuchar, and Katherine E. Wolf. Exploring the analytical processes of intelligence analysts. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 11–20, 2009.

[14] Laura Chiticariu, Phokion G. Kolaitis, and Lucian Popa. Interactive generation of integrated schemas. In *ACM SIGMOD*, pages 833–846, 2008.

[15] Peter Christen. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 1065–1068, New York, NY, USA, 2008. ACM.

[16] C.D. Correa, Yu-Hsuan Chan, and Kwan-Liu Ma. A framework for uncertainty-aware visual analytics. In *IEEE Visual Analytics Science and Technology*, pages 51–58, Oct 2009.

[17] D. Chappell. The workflow way: Understanding windows workflow foundation, white paper.

[18] Catalina M. Danis, Fernanda B. Viégas, Martin Wattenberg, and Jesse Kriss. Your place or mine?: visualization as a community component. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 275–284, 2008.

[19] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., 2003.

[20] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *ACM SIGMOD*, pages 240–251, 2002.

[21] Cyntrica Eaton, Catherine Plaisant, and Terence Drizd. The challenge of missing and uncertain data. In *VIS '03: 14th IEEE Visualization 2003 (VIS'03)*, page 100, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Wayne Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. Technical report, The Data Warehousing Institute, 2002.

[23] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowledge & Data Engineering*, 19(1):1–16, 2007.

[24] G.A. Fink, C.L. North, Alex Endert, and S. Rose. Visualizing cyber security: Usable workspaces. In *Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on*, pages 45 –56, 2009.

[25] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. Trust me, I'm partially right: Incremental visualization lets analysts explore large datasets faster. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 1673–1682, 2012.

[26] Kathleen Fisher and David Walker. The Pads project: An overview. In *International Conference on Database Theory*, March 2011.

[27] Juliana Freire, David Koop, Emanuele Santos, and Claudio T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10:11–21, 2008.

[28] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. In *ACM SIGMOD*, page 590, 2000.

[29] John F. Gantz, Christopher Chute, Alex Manfrediz, Stephen Minton, David Reinsel, Wolfgang Schlichting, and Anna Toncheva. The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011. An idc white paper - sponsored by emc, IDC, March 2008.

[30] Nahum D. Gershon. Visualization of fuzzy data using generalized animation. In *Proceedings of the 3rd conference on Visualization '92*, VIS '92, pages 268–273, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

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

[32] Henning Griethe and Heidrun Schumann. The visualization of uncertain data: Methods and problems. In *SimVis*, pages 143–156, 2006.

[33] G. Grigoryan and P. Rheingans. Point-based probabilistic surfaces to show surface uncertainty. *Visualization and Computer Graphics, IEEE Transactions on*, 10(5):564 –573, 2004.

[34] Diansheng Guo. Coordinating computational and visual approaches for interactive feature selection and multivariate clustering. *Information Visualization*, 2(4):232–246, 2003.

[35] Philip J. Guo, Sean Kandel, Joseph Hellerstein, and Jeffrey Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *ACM User Interface Software & Technology (UIST)*, 2011.

[36] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD*, pages 805–810, 2005.

[37] Jeffrey Heer and Maneesh Agrawala. Design considerations for collaborative visual analytics. *Information Visualization*, 7:49–62, 2008.

[38] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Trans. Visualization & Computer Graphics (Proc. InfoVis)*, 14:1189–1196, 2008.

[39] 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*, 14:1189–1196, 2008.

[40] Jeffrey Heer, Fernanda Viegas, and Martin Wattenberg. Voyagers and voyeurs: Supporting asynchronous collaborative information visualization. In *ACM CHI*, pages 1029–1038, 2007.

[41] Joseph M. Hellerstein. Quantitative data cleaning for large databases, 2008. White Paper, United Nations Economic Commission for Europe.

[42] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *Proc. ACM SIGMOD*, pages 171–182, 1997.

[43] Richard J. Heuer. *Psychology of Intelligence Analysis*. Center for the Study of Intelligence, 1999.

[44] A. Hinneburg, D.A. Keim, and M. Wawryniuk. HD-Eye: visual mining of high-dimensional data. *IEEE CG&A*, 19(5):22 –31, 1999.

[45] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[46] Eric Horvitz. Principles of mixed-initiative user interfaces. In *ACM CHI*, pages 159–166, 1999.

[47] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[48] David Huynh and Stefano Mazzocchi. Google Refine. `http://code.google.com/p/google-refine/`.

[49] David Huynh and Stefano Mazzocchi. Google Refine. http://code.google.com/p/google-refine/.

[50] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: semi-ontology alignment for casual users. In *ISWC*, pages 903–910, 2007.

[51] Robert Ikeda, Akash Das Sarma, and Jennifer Widom. Logical provenance in data-oriented workflows. In *ICDE 2013*. Stanford InfoLab.

[52] P. Isenberg, D. Fisher, M.R. Morris, K. Inkpen, and M. Czerwinski. An exploratory study of co-located collaborative visual analytics around a tabletop display. In *Proc. IEEE Visual Analytics Science and Technology (VAST)*, pages 179–186, 2010.

[53] Petra Isenberg, Anthony Tang, and Sheelagh Carpendale. An exploratory study of visual information analysis. In *Proc. ACM Human factors in Computing Systems (CHI)*, pages 1217–1226, 2008.

[54] Zachary G. Ives, Craig A. Knoblock, Steven Minton, Marie Jacob, Partha Pratim, Talukdar Rattapoom Tuchinda, Jose Luis, Ambite Maria, and Muslea Cenk Gazen. Interactive data integration through smart copy & paste. In *Proc. CIDR*, 2009.

[55] T.J. Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz. A model and framework for visualization exploration. *IEEE TVCG*, 13:357–369, 2007.

[56] 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*, 10:271–288, 2011.

[57] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proc. ACM CHI*, pages 3363–3372, 2011.

[58] Hyunmo Kang, Lise Getoor, Ben Shneiderman, Mustafa Bilgic, and Louis Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE Trans. Visualization & Computer Graphics*, 14(5):999–1014, 2008.

[59] Y. Kang, C. Görg, and J. Stasko. Evaluating visual analytics systems for investigative analysis: Deriving design principles from a case study. In *Proc. IEEE Visual Analytics Science and Technology (VAST)*, pages 139–146, 2009.

[60] Y. Kang and J. Stasko. Characterizing the intelligence analysis process: Informing visual analytics design through a longitudinal field study. In *Proc. IEEE Visual Analytics Science and Technology (VAST)*, pages 21–30, 2011.

[61] D.A. Keim. Information visualization and visual data mining. *IEEE TVCG*, 8(1):1–8, 2002.

[62] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. A taxonomy of dirty data. *Data Mining & Knowl. Discovery*, 7(1):81–99, 2003.

[63] Robert Kosara. *Semantic Depth of Field âĂŞ Using Blur for Focus+Context Visualization*. PhD thesis, Vienna University of Technology, Austria, 2001.

[64] Robert Kosara, Fabian Bendix, and Helwig Hauser. TimeHistograms for large, time-dependent data. In *Proc. VisSym*, pages 45–54, 2004.

[65] M. Kreuseler, T. Nocke, and H. Schumann. A history mechanism for visual data mining. In *IEEE InfoVis*, pages 49–56, 2004.

[66] Bumchul Kwon, Brian Fisher, and Ji Soo Yi. Visual analytic roadblocks for novice investigators. In *Proc. IEEE Visual Analytics Science and Technology (VAST)*, pages 3–11, 2011.

[67] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Trans. Database Syst.*, 26(4):476–519, 2001.

[68] Bongshin Lee, George G. Robertson, Mary Czerwinski, and Cynthia Sims Parr. Candidtree: visualizing structural uncertainty in similar hierarchies, December 2007.

[69] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A. Lau. End-user programming of mashups with Vegemite. In *Proc. Intelligent User Interfaces*, pages 97–106, 2009.

[70] Suresh K. Lodha, Catherine M. Wilson, and Robert E. Sheehan. Listen: sounding uncertainty visualization. In *Proceedings of the 7th conference on Visualization '96*, VIS '96, pages 189–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[71] 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, May 2011.

[72] William K. Michener, James H. Beach, Matthew B. Jones, Bertram Ludäscher, Deana D. Pennington, Ricardo S. Pereira, Arcot Rajasekar, and Mark Schildhauer. A knowledge environment for the biodiversity and ecological sciences. *J. Intell. Inf. Syst.*, 29:111–126, August 2007.

[73] Robert C. Miller and Brad A. Myers. Interactive simultaneous editing of multiple text regions. In *USENIX Tech. Conf.*, pages 161–174, 2001.

[74] Bonnie A. Nardi and James R. Miller. An ethnographic study of distributed problem solving in spreadsheet development. pages 197–208. ACM Press, 1990.

[75] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 2002.

[76] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coodinating visualizations via relational schemata. In *Proc. Advanced Visual Interfaces*, pages 128–135, 2000.

[77] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[78] Carl W. Olofson and Dan Vesset. Worldwide Hadoop-MapReduce ecosystem software 2012–2016 forecast. May 2012.

[79] C. Olston and J. Mackinlay. Visualizing data with bounded uncertainty. In *IEEE Symposium on Information Visualization*, pages 37–40, Boston, Massachusetts, October 2002.

[80] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proc. ACM SIGMOD*, pages 1099–1110, 2008.

[81] T. Pang, C. M. Wittenbrink, and S. K. Lodha. Approaches to uncertainty visualization. In *The Visual Computer*, pages 370–390, Nov 1997.

[82] Adam Perer and Ben Shneiderman. Systematic yet flexible discovery: guiding domain experts through exploratory data analysis. In *Proc. Intelligent User interfaces*, pages 109–118, 2008.

[83] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proc. International Conference on Intelligence Analysis*, 2005.

[84] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.

[85] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23, 2000.

[86] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[87] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[88] George G. Robertson, Mary P. Czerwinski, and John E. Churchill. Visualization of mappings between schemas. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 431–439, 2005.

[89] Geraldine E. Rosario, Elke A. Rundensteiner, David C. Brown, Matthew O. Ward, and Shiping Huang. Mapping nominal values to numbers for effective visualization. *Information Visualization*, 3(2):80–95, 2004.

[90] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. The cost structure of sensemaking. In *Proc. ACM Human Factors in Computing Systems (CHI)*, pages 269–276, 1993.

[91] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *ACM SIGKDD*, Edmonton, Alberta, 2002.

[92] Christopher Scaffidi, Brad Myers, and Mary Shaw. Intelligently creating and recommending reusable reformatting rules. In *Proc. Intelligent User Interfaces*, pages 297–306, 2009.

[93] Michael Sedlmair, Petra Isenberg, Dominikus Baur, and Andreas Butz. Evaluating information visualization in large companies: Challenges, experiences and recommendations. In *Proc. CHI Workshop Beyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV)*, 2010.

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

[95] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[96] Yedendra Babu Shrinivasan and Jarke van Wijk. Supporting exploration awareness in information visualization. *IEEE Computer Graphics and Applications*, 29:34–43, 2009.

[97] Meredith Skeels, Bongshin Lee, Greg Smith, and George G. Robertson. Revealing uncertainty for information visualization. *Information Visualization*, 9:70–81, March 2010.

[98] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, 1999.

[99] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explorations Newsletter*, 1(2):12–23, January 2000.

[100] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE TVCG*, 8(1):52–65, 2002.

[101] Deborah F Swayne, Duncan Temple Lang, Andreas Buja, and Dianne Cook. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Comp. Stat. & Data Analysis*, 43(4):423–444, 2003.

[102] Ruey S. Tsay. Outliers, level shifts, and variance changes in time series. *Journal of Forecasting*, 7(1):1–20, 1988.

[103] Rattapoom Tuchinda, Pedro Szekely, and Craig A. Knoblock. Building mashups by example. In *Proc. Intelligent User Interfaces*, pages 139–148, 2008.

[104] J. Cavallo Twiddy and S. M. Shiri. Restorer: A visualization technique for handling missing data. In *IEEE Visualization*, pages 212–216, 2004.

[105] Antony Unwin, Martin Theus, and Heike Hofmann. *Graphics of Large Datasets: Visualizing a Million*. Springer, 2006.

[106] Fernanda B. Viegas, Martin Wattenberg, Matt McKeon, Frank van Ham, and Jesse Kriss. Harry potter and the meat-filled freezer: A case study of spontaneous usage of visualization tools. In *HICSS '08: 41st Annual Hawaii International Conference on System Sciences*, page 159, Washington, DC, USA, 2008. IEEE Computer Society.

[107] Martin Wattenberg and Jesse Kriss. Designing for social data analysis. *IEEE Trans. Visualization & Computer Graphics*, 12(4):549–557, July 2006.

[108] C. Weaver. Building highly-coordinated visualizations in Improvise. In *Proc. IEEE InfoVis*, pages 159–166, 2004.

[109] Qian Xi, Kathleen Fisher, David Walker, and Kenny Q. Zhu. Ad hoc data and the token ambiguity problem.

[110] J. Yang, M. O. Ward, E. A. Rundensteiner, and S. Huang. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *Proc. VisSym*, pages 19–28, 2003.