Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment

Sean Kandel*, Ravi Parikh*, Andreas Paepcke*, Joseph M. Hellerstein†, Jeffrey Heer*
*Stanford University †University of California, Berkeley
{skandel, rparikh, paepcke, jheer}@cs.stanford.edu, hellerstein@cs.berkeley.edu

ABSTRACT

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.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

Data analysis, visualization, data quality, anomaly detection

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 [6]. 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 [5].

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.

2. RELATED WORK

Profiler draws on three areas of related work: anomaly detection, data cleaning tools, and visual analysis systems.

2.1 Classifying Data Anomalies

The database and statistics literature includes many taxonomies of anomalous data [5, 7, 10, 19, 24, 33]. These taxonomies inform

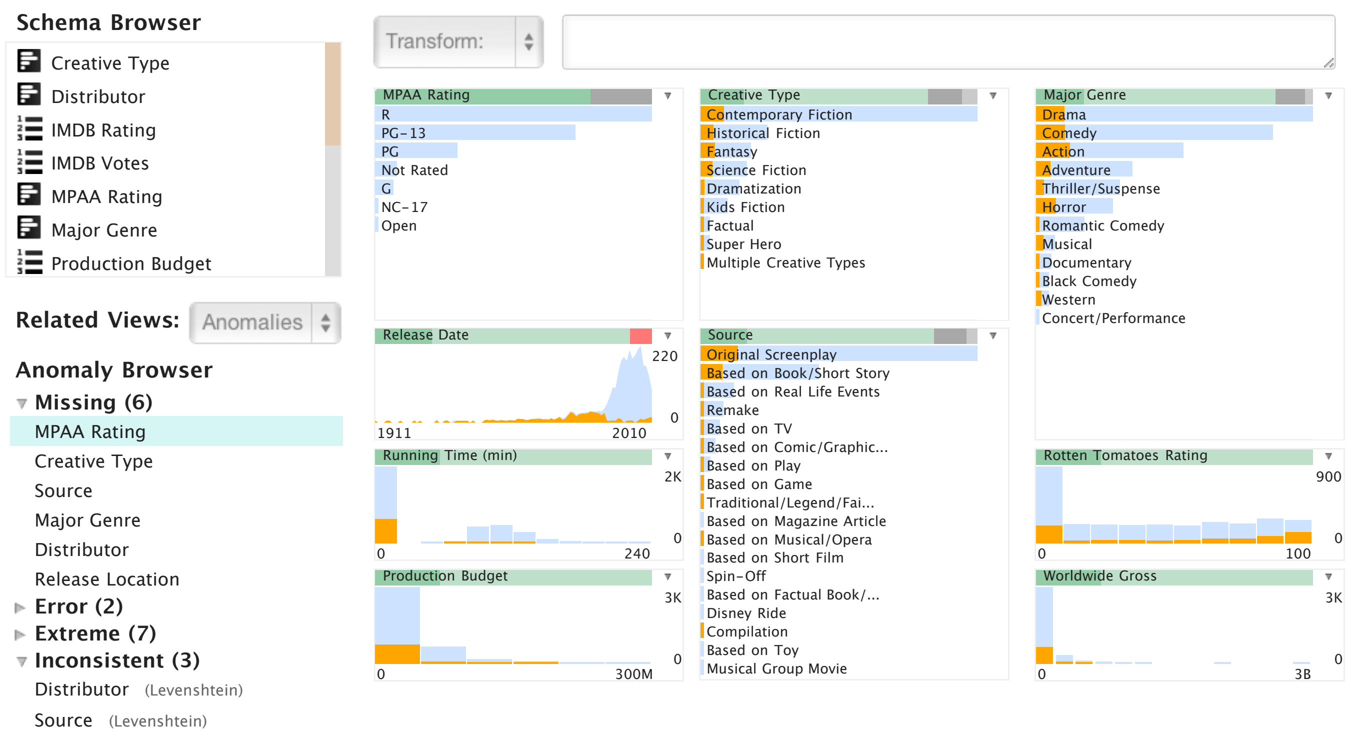


Figure 1: The Profiler User Interface. The UI contains (clockwise from top-left): (a) schema browser, (b) formula editor, (c) canvas of linked summary visualizations, and (d) anomaly browser. Profiler generates a set of linked views for each identified anomaly. Here, we investigate possible causes of 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.

a variety of algorithms for detecting outliers [4, 10, 12, 33], duplicate records [7], and key violations [13]. While these routines flag potential issues, most types of error require some form of human intervention to assess and correct [19]. Here, 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 [1].

Erroneous data can arise because of error during data entry, measurement, or distillation [10]. 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 [33]: 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).

2.2 Data Cleaning Tools

Motivated by the issues above, database and HCI researchers have created interactive systems for data cleaning. Many of these interfaces focus on data integration [9, 15, 21, 27, 34] or entity resolution [17]. Here we focus on data quality issues in a single table. Profiler does include detectors for duplicate values, but we do not attempt to address the general problem of entity resolution.

Other interfaces support mass reformatting of raw input data [14, 16, 25, 29]. A common form of discrepancy detection is provided by data type definitions that specify constraints for legal values [16, 25, 29]. These systems are usually limited to finding formatting discrepancies for individual values. Profiler's data types are similar to *domains* in Potter's Wheel [25] and Scaffidi et al.'s *Topes* [29]. 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.

Perhaps most comparable to Profiler is Google Refine [14], which 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.

Profiler is integrated with the Wrangler [16] data transformation tool. An analyst can transform raw data using Wrangler. Once the data is properly formatted as a relational table, Profiler can leverage type information to automate anomaly detection and visualization.

2.3 Visual Analysis Systems

Visualization can support discovery of patterns in data, including anomalies [18]. Aggregation, clustering and sorting have been used in various contexts to support scalable visualization for large data sets [3, 20, 28, 35]. Through linked highlighting (“brushing & linking”), coordinated multiple views enable assessment of relationships between data dimensions [22, 36]. Profiler’s visualization layer extends this prior work with a set of type-specific aggregate visualizations that aid assessment of data quality issues.

Visual analytic tools such as Tableau [31], GGobi [32], and Improvise [36] 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 [23] uses a wizard-like interface to guide users through social network analysis. Seo and Shneiderman’s rank-by-feature framework [30] 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 [8, 11, 37]. 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.

3. 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 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 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 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.

The analyst next decides to look at extreme values in financial figures and clicks Worldwide Gross in the ‘Extreme’ anomaly list.

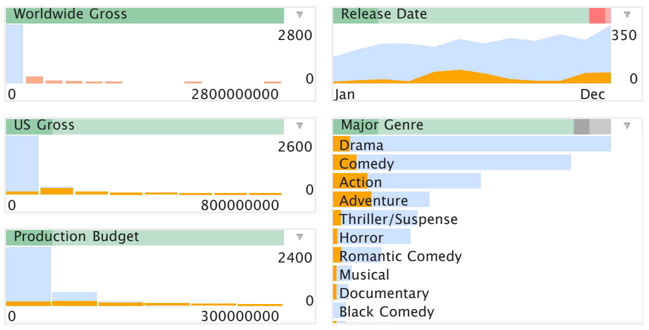


Figure 2: Automatically generated views to help assess Worldwide Gross. Worldwide Gross correlates with high US Gross and Production Budgets. High gross also coincides with Action & Adventure movies and the Summer & Winter seasons. Profiler chose to bin Release Date by month instead of by year.

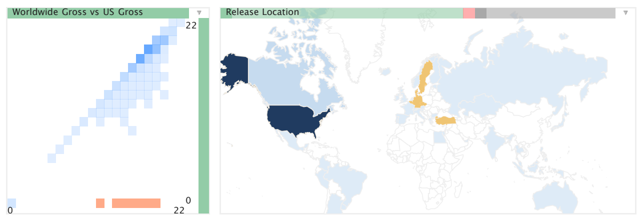
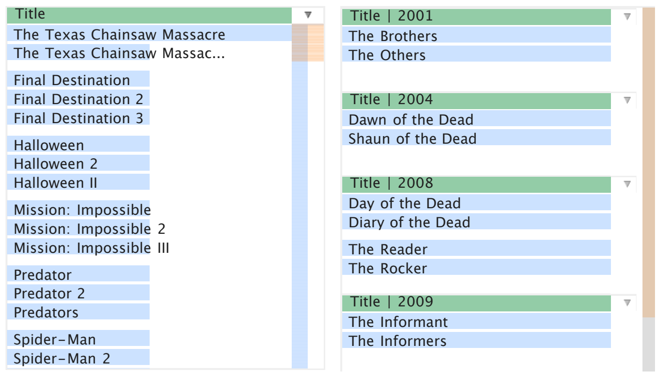


Figure 3: Map assessing 2D outliers in a binned scatter plot. Selected in the scatter plot are movies with high Worldwide Gross but low US Gross (in orange). Linked highlights on the map confirm that the movies were released outside of the US.



Title
The Texas Chainsaw Massacre
The Texas Chainsaw Massac...

Final Destination
Final Destination 2
Final Destination 3

Halloween
Halloween 2
Halloween II

Mission: Impossible
Mission: Impossible 2
Mission: Impossible III

Predator
Predator 2
Predators

Spider-Man
Spider-Man 2

Title | 2001
The Brothers
The Others

Title | 2004
Dawn of the Dead
Shaun of the Dead

Title | 2008
Day of the Dead
Diary of the Dead

The Reader
The Rocker

Title | 2009
The Informant
The Informers

Figure 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.

A histogram reveals a small number of high 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 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 de-clutter 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 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 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.

4. 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 [16] 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 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.

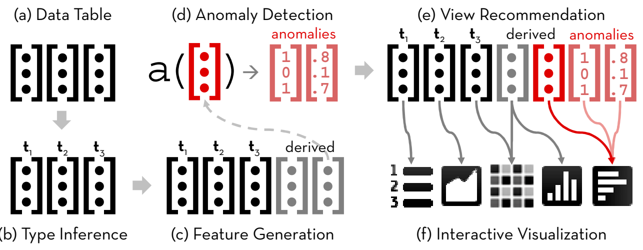


Figure 5: The Profiler Architecture. An (a) input table is analyzed to (b) infer types for each column. Type information is used to (c) generate features prior to running (d) anomaly detection routines. The results of anomaly detection and mutual information analysis are used to perform (e) view recommendation and populate a set of (f) interactive visualizations.

4.1 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 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 based on the Minimum Description Length principle (MDL) [26]. 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 [25]. We use the same MDL formulation in Profiler.

4.2 Detector

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

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: 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.

4.2.1 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.

4.2.2 Detection Routines

Profiler incorporates a variety of detection routines to flag data anomalies (Figure 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 (Sec. 4.1). 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 [7]. 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 [10].

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 [10]. 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 [10]. 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.

4.3 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.

4.3.1 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:

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

4.3.2 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-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 augment R to include larger subsets of columns.

4.4 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 [2]. We now detail the design of the View Manager, including optimizations for rendering and query performance.

4.4.1 Summary Visualizations

Visualizing "raw" data is increasingly difficult with even moderately sized data — even a few hundred data points may make a

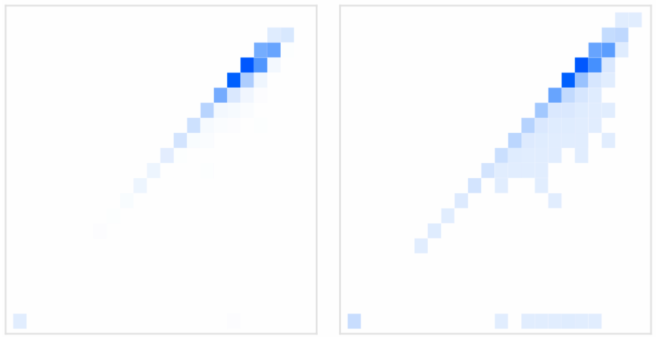


Figure 7: Adding perceptual discontinuity to summary views. Left: A binned scatter plot using a naive opacity ramp from 0-1. Right: An opacity scale with a minimum non-zero opacity ensures perception of bins containing relatively few data points.

scatter plot difficult to read due to overplotting. Profiler's summary visualizations use aggregation to scale to a large number of records [3, 14, 20, 28, 35]: 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 [3] 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 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.

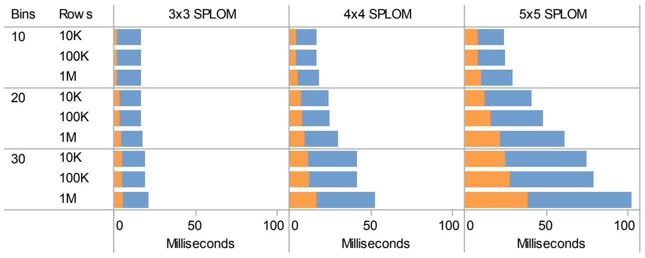


Figure 8: Performance (in ms) of linked highlighting in a scatter plot matrix (SPLOM). Orange bars represent query processing time, blue bars represent rendering time. We varied the number of dimensions, bins per dimension and data set size. In most cases we achieve interactive (sub-100ms) response rates.

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.

4.4.2 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 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 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.

4.4.3 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 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.

5. 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. CONCLUSION

In this paper 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.

In future work, we plan to evaluate Profiler through both controlled studies and public deployments on the web. We intend to develop a tool for end users to define custom types (c.f., [29]) and to incorporate detectors and visualizations for additional data types such as free-form text. Our 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 side-step redundancy and further improve view ranking.

7. ACKNOWLEDGEMENTS

The first author was supported by a Stanford Graduate Fellowship. This work was partially funded by NSF grant CCF-0964173.

8. REFERENCES

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

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

[3] 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.

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

[5] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., 2003.

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

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.

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

[9] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD*, pages 805–810, 2005.

[10] J. M. Hellerstein. Quantitative data cleaning for large databases, 2008. White Paper, U.N. Economic Commission for Europe.

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

[12] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Rev.*, 22(2):85–126, 2004.

[13] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[14] D. Huynh and S. Mazzocchi. Google Refine. http://code.google.com/p/google-refine/.

[15] Z. G. Ives, C. A. Knoblock, S. Minton, M. Jacob, P. Pratim, T. R. Tuchinda, J. Luis, A. Maria, and M. C. Gazen. Interactive data integration through smart copy & paste. In *Proc. CIDR*, 2009.

[16] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proc. ACM CHI*, pages 3363–3372, 2011.

[17] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. Interactive entity resolution in relational data: A visual analytic tool and its evaluation. *IEEE TVCG*, 14(5):999–1014, 2008.

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

[19] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data Mining & Knowl. Discovery*, 7(1):81–99, 2003.

[20] R. Kosara, F. Bendix, and H. Hauser. TimeHistograms for large, time-dependent data. In *Proc. VisSym*, pages 45–54, 2004.

[21] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with Vegemite. In *Proc. Intelligent User Interfaces*, pages 97–106, 2009.

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

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

[24] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23, 2000.

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

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

[27] G. G. Robertson, M. P. Czerwinski, and J. E. Churchill. Visualization of mappings between schemas. In *Proc. ACM CHI*, pages 431–439, 2005.

[28] G. E. Rosario, E. A. Rundensteiner, D. C. Brown, M. O. Ward, and S. Huang. Mapping nominal values to numbers for effective visualization. *Information Visualization*, 3(2):80–95, 2004.

[29] C. Scaffidi, B. Myers, and M. Shaw. Intelligently creating and recommending reusable reformatting rules. In *Proc. Intelligent User Interfaces*, pages 297–306, 2009.

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

[31] 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.

[32] D. F. Swayne, D. T. Lang, A. Buja, and D. Cook. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Comp. Stat. & Data Analysis*, 43(4):423–444, 2003.

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

[34] R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. In *Proc. Intelligent User Interfaces*, pages 139–148, 2008.

[35] A. Utwin, M. Theus, and H. Hofmann. *Graphics of Large Datasets: Visualizing a Million*. Springer, 2006.

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

[37] 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.