Efficient Visualization Recommendation under Updates

Todd Yu
Aditya Parameswaran, Ed.
Dixin Tang, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-116
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-116.html

May 11, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I'd like to thank Dr. Dixin Tang for his unwavering mentorship and support in this work, as well as Professor Aditya Parameswaran for his insightful guidance and patience, and Devin Petersohn and Rehan Durrani for their support and nurturing. These figures have been instrumental in my journey as an undergraduate and M.S. student, and I would not have found research nearly as enjoyable or meaningful without them. Truly, from the bottom of my heart, I thank you for helping me grow.

I'd like to also thank EPIC Data Lab and RISE/Sky Lab for providing resources such as compute credits and lab space.

Efficient Visualization Recommendation under Updates
by Todd Yu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Aditya Parameswaran
Research Advisor

5/11/2023

(Date)

* * * * * * *

Dr. Dixin Tang
Second Reader

05/11/2023

(Date)

Efficient Visualization Recommendation under Updates

by

Todd Yu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master's of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Aditya Parameswaran, Chair
Dr. Dixin Tang

Spring 2023

Efficient Visualization Recommendation under Updates

Copyright 2023
by
Todd Yu

Abstract

Efficient Visualization Recommendation under Updates

by

Todd Yu

Master's of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Aditya Parameswaran, Chair

Visualization Recommendation (VisRec) systems are popular for generating visualizations with limited effort during the Exploratory Data Analysis (EDA) process. However, computing visualization recommendations can be slow. A high proportion of computation time involves calculating ranking scores (i.e., statistical utility metrics based on underlying data) for a wide range of possible visualizations. In this work, our goal is to incrementally maintain VisRec ranking scores in EDA workflows under data updates. Updates to data are common, thanks to user-driven data cleaning or transformation steps, as well as external data updates. Our primary challenges stem from analyzing a wide range of VisRec systems and their ranking scores, as well as covering a broad set of data updates, to identify how to maintain updates to scores incrementally. We must also determine, upon updates, when to incrementally maintain ranking scores and when to recompute them from scratch. We first review an existing taxonomy of common VisRec categories, known as analytical actions, then decompose all visualization ranking scores into a minimal set of five core aggregates per column. We then propose a system to maintain these aggregates incrementally by presenting five core operators for tabular data that compose to make up a wide variety of common EDA data transformations, and then showing how we can efficiently update our ranking scores for these operations. We also propose a cost model to determine when to incrementally update ranking scores, and when to recompute scores from scratch. We implement our approach in Lux, a popular open-source VisRec system for pandas dataframes, and show how our system scales. We demonstrate the efficacy of our cost model by showing that for datasets with many columns and thus many ranking scores, our system is faster or equivalent to naive recomputation upon updates.

To my mother, who believed in me first.

Contents

Contents **ii**

List of Figures **iv**

List of Tables **vi**

1 Introduction **1**
1.1 Visual Recommendation Systems 1
1.2 The Challenges with Updates 3
1.3 Research Goals and Contributions 4
1.4 Related Work 6

2 Overview of Approach **7**
2.1 Defining Ranking Score Aggregates and Core Operators 8
2.2 Efficiently Maintaining Ranking Scores 8

3 VisRec System Ranking Scores **9**
3.1 VisRec System Taxonomy Overview 9
3.2 Defining Existing Ranking Scores and Aggregates 11
3.3 Decomposing Ranking Scores 17
3.4 Limitations 18

4 Maintaining VisRec Ranking Score Aggregates **19**
4.1 Tabular Data Operations and Maintaining VisRec Scores 19
4.2 Cost Model for Updates 23
4.3 Limitations 26

5 Implementation and Performance Evaluation **27**
5.1 Implementation in Lux 27
5.2 Dataframe Use Case Examples 28
5.3 Evaluation 28

6 Conclusion and Future Work **31**

6.1 Conclusion 31
6.2 Future Work 31

Bibliography **33**

List of Figures

1.1 A taxonomy of common visual recommendation analytical actions, created by Lee et al. [14]. The final analytical action categories are to the right, highlighted in blue. 2
1.2 Diagram of Exploratory Data Analysis (EDA) Workflow with a VisRec system, where a data scientist repeatedly invokes the system after each step. 3
1.3 Example of Lux, a VisRec system, operating in a Jupyter Notebook Environment. The "college" dataset is rapidly ingested into a dataframe and then profiled and visualized with Lux in an eager fashion and in quick succession, with little to no user code. 5

2.1 Illustrating our proposed approach for incrementally maintaining ranking scores. Common EDA operations can be composed of core tabular data operators, each of which efficiently update a set of core ranking score aggregates. These aggregates are used to efficiently derive common VisRec ranking scores. 7

4.1 Existing VisRec systems such as Lux [15] and AutoViz [3] compute ranking scores upon visualization, but recompute ranking scores from scratch upon updates, as shown in Figure 4.1. 20
4.2 A dataframe, an example of a tabular data model, with row and column labels. A dataframe with N rows and M columns will have $N \cdot M$ cells in "Array of Data". 21
4.3 We incrementally maintain VisRec ranking score aggregates upon data updates, instead of fully recomputing statistics from scratch as seen in Figure 4.1. 22
4.4 A representation graph displaying quantitative column X. X is modeled as a source, with directed edges to its aggregates $\sum_i X_i$, $\sum_i X_i^2$, $|X|$, and $\sum_i X_i Y_i$, as well as its downstream, filtered column X_F. Note that Y, an additional column, and X_F also have their own dependencies (aggregates to maintain), modeled as additional connected nodes. 24

5.1 Within the Airbnb dataset, we apply a fixed number of 10000 operations (5000 row deletes and 5000 row additions), while fixing the number of columns (16) and increasing the number of rows from 250,000 to 2 million. Updating ranking scores takes a fixed amount of time regardless of number of rows. However, for smaller datasets, full recomputation is faster. We demonstrate in the next section that when the number of columns in a dataset increases (and thus the number of possible visualizations and ranking scores increases), our strategy becomes much more effective. 29
5.2 We compare efficacy of the incremental aggregate update strategy versus full statistic/ranking score recomputation in Lux on the Crime Data dataset (2000 rows, 128 columns), varying the number of row additions and deletes from 6.25% of the number of rows (125 rows) to 150% of the number of rows (3000 rows). We see that for row operations up to 100% of the number of rows (2000 rows), incrementally updating aggregates is preferred. For any more updates, full ranking score recomputation from scratch is faster, as demonstrated by our cost model. . 30

List of Tables

3.1 Decomposing ranking scores for Data Variable (Column) X and (optional) external column Y into their respective aggregates. The columns of the table are listed as follows: sum of X, sum of Y, sum of X's squared elements, sum of Y's squared elements, inner product of X and Y, cardinality of X. $\gamma_X = V_X$ is values over an aggregate over X, X_F represents filtered values, and V_{flat} represents the flat uniform distribution based on cardinality. 17
3.2 Aggregates to maintain for Column X and external Column Y. 18

Acknowledgments

I'd like to thank Dr. Dixin Tang for his unwavering mentorship and support in this work, as well as Professor Aditya Parameswaran for his insightful guidance and patience, and Devin Petersohn and Rehan Durrani for their support and nurturing. These figures have been instrumental in my journey as an undergraduate and M.S. student, and I would not have found research nearly as enjoyable or meaningful without them. Truly, from the bottom of my heart, I thank you for helping me grow.

I'd like to also thank EPIC Data Lab and RISE/Sky Lab for providing resources such as compute credits and lab space.

Chapter 1

Introduction

1.1 Visual Recommendation Systems

1.1.1 Introduction to Visual Recommendation (VisRec) Systems

The modern Exploratory Data Analysis (EDA) process consists of an iterative process of asking questions, visualizing data, and performing data processing and transformation operations. Data visualization is a core component of the EDA process; it allows data scientists to ask thoughtful questions and reason about data, as well as examine underlying trends and patterns.

Visual Recommendation (VisRec) systems have evolved to aid the visual data analytics process by not only drastically reducing tedious effort in processing and inspecting data, but also suggesting insightful visual encodings or potentially interesting visualizations. VisRec systems serve a variety of functions: automatically processing and visualizing data, recommending visualizations, and allowing users to easily specify and materialize a collection of visualizations. Many VisRec systems such as Lux, SeeDB, AutoViz, and DIVE [15, 29, 25, 33, 32, 3, 13, 11] also offer low-code/no-code interfaces, making them extremely versatile. Some systems are mixed-initiative [25, 15, 32, 27], supporting both user specifications and automatically suggesting interesting visualizations. Others are entirely automatic [30]. Many operate on tabular data models, organized into columns for data fields and rows for individual data entries [15, 3, 29, 25, 11].

1.1.2 Recommendation Categories and Analytical Actions

Visual recommendations generated by a VisRec system can be grouped into categories based on specific analytical actions corresponding to various forms of exploration: unit operations that can be performed by the user during the analysis process. Examples of analytical actions include visualizing distributions of various columns or plotting correlation between two data variables of interest [14, 12, 23]. Lee et al. formalized a taxonomy of common analytical action-based recommendation categories for data-based VisRec systems (systems that use

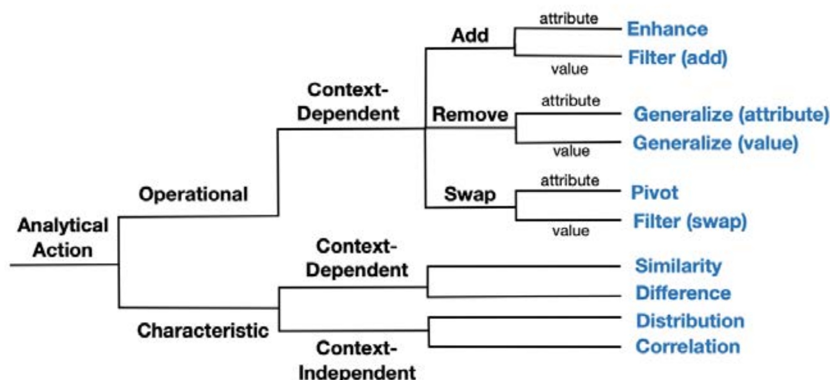

Figure 1.1: A taxonomy of common visual recommendation analytical actions, created by Lee et al. [14]. The final analytical action categories are to the right, highlighted in blue.

statistical properties of the underlying data to identify visualization recommendations) [14], shown in Figure 1.1. Actions are first grouped into *operational* and *characteristic* categories: *operational* actions correspond to those that are user-invoked (such as adding a filter or additional data variable to the current view), while *characteristic* actions reveal characteristic patterns in data such as correlations. Actions are further broken down into whether they are *context-dependent*–if they depend on the current view/visualization–or *context-independent* otherwise. While the VisRec design space is extremely vast and constantly evolving, we use Lee et al.'s taxonomy, synthesized via a thorough review of papers on this space, to systematically enumerate many common data-based VisRec system actions and use cases for the challenges outlined in the following sections. We elaborate on other categorization frameworks, as well as justify our decision to use Lee et al.'s taxonomy, in the Related Work section of this chapter and Chapter 2.

1.1.3 Visualization Generation and Ranking

VisRec systems [25, 33, 32, 29, 31, 13, 15, 11] often use ranking scores to measure the value of a visualization to the end user, also known as "interestingness" [14]. Ranking scores can be assigned for visualizations within a particular action category, or for visualizations across action categories. Examples of interestingness scores include distance metrics (Euclidean distance, Earth mover's distance, or Jenson-Shannon Divergence), correlation, skewness, separability, and more. For instance, analysts may be more interested in bar charts that are heavily skewed, or scatterplots that exhibit high degrees of correlation. These visualizations should therefore be ranked higher in priority by VisRec systems to be displayed to the user. We define a set of ranking scores for commonly-occurring basic visualization types including

bar charts, histograms, line charts, and scatterplots in Chapter 3, and show how to maintain them efficiently with respect to updates in Chapter 4.

1.2 The Challenges with Updates

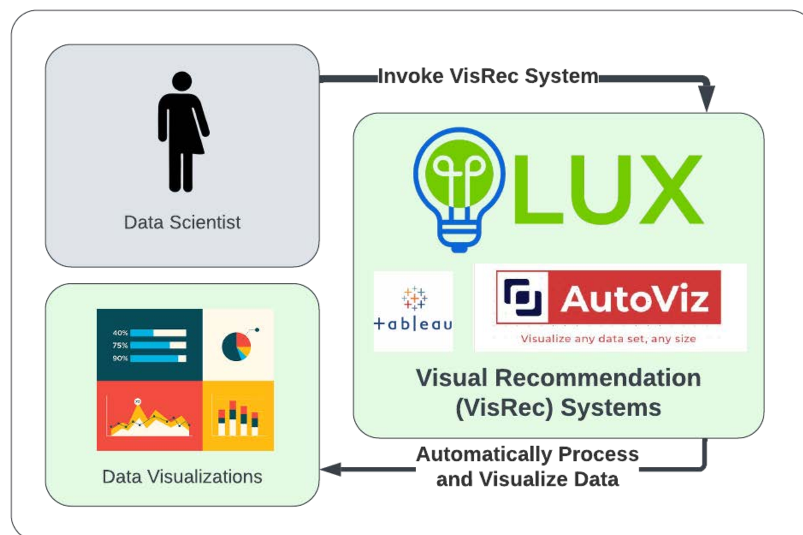

Figure 1.2: Diagram of Exploratory Data Analysis (EDA) Workflow with a VisRec system, where a data scientist repeatedly invokes the system after each step.

1.2.1 VisRec Systems within the EDA Process

Users visualize, clean, and explore data iteratively during the EDA process. VisRec systems are iteratively invoked as data is repeatedly cleaned, processed and transformed, then re-visualized; this process is illustrated in Figure 1.2. As an example, a data scientist may visualize data, then clean data by imputing (filling) null values or applying a filter to specific rows, and then re-visualize the data, as one step of the EDA process.

Repeated invocation of VisRec systems is often a computationally expensive but important step in deriving data insights with maximum utility. However, VisRec systems are not optimized for computation under repeated data updates in the EDA process [15, 29, 13, 3]; they naively recompute results after data updates are performed. Much of the computation time comes from computing visualization ranking scores (statistical measures of effectiveness) for candidate visualizations. And in many cases, **small data modifications may lead to expensive full recomputation** across a large dataset.

As a case study, Agarwal et al. [1] profiled Lux, a VisRec system for dataframes with 4.5k GitHub stars as of March 2023. Agarwal et al. found that Lux's full computation cycle can take **over 30 seconds** on "tall" datasets of approximately 4 million rows and 16 columns, and **over 200 seconds** on "wide" datasets of 128 columns and 500,000 rows. Agarwal's detailed profiling identifies **statistics and metadata collection as being the primary bottleneck for runtime**, with costs for collecting statistics per column and ranking visualizations based on "interestingness" (ranking score) dominating over 70% of the computation time in both examples. Lee et al.'s user study on Lux [15] reported 12/16 participants being *most interested in improving Lux's latency on large datasets* out of any suggested future revisions, even though users also reported Lux dramatically speeding up the EDA process overall (by almost two-fold for some participants).

Existing VisRec systems focus on computing statistics and metadata [3, 15, 11], but no methods we are aware of emphasize maintaining these statistics/metadata, specifically ranking scores, under data updates. **We aim to speed up computation in VisRec systems by efficiently maintaining ranking scores under data updates**.

1.2.2 Many types of VisRec Systems and Data Updates

Many VisRec systems exist [3, 11, 15, 29, 13, 33, 32, 27, 25], and each has a variety of visualization objectives and possible actions. The space of possible visualizations and associated ranking scores is vast. We use Lee et al.'s VisRec system taxonomy to map the VisRec system design space and focus our attention on common VisRec system actions/visualizations. In addition, the space of possible data updates is extremely broad. **Our challenge is to consider a variety of recommendation categories and updates.** We limit our scope to tabular data models (employed by many VisRec systems [15, 3, 13, 33, 29, 27, 25]), and propose a set of five core operators that update tabular data in Chapter 3. We provide an overview of our strategy in Chapter 2.

In addition, another key challenge stems from determining when to maintain ranking scores under data updates, and when to recompute ranking scores from scratch. We develop a cost model in Chapter 4 to estimate when our strategy of maintaining ranking sores will be computationally cheaper than fully recomputing ranking scores from scratch.

1.3 Research Goals and Contributions

We aim to **reduce user-facing latency in VisRec systems under data updates**. To address the challenges mentioned above, we must cover a wide range of VisRec systems and their ranking scores, as well as cover a broad set of data updates. We must also determine, upon specific data updates, when to incrementally maintain ranking scores, and when full recomputation is more efficient. We outline our approach in greater detail in Chapter 2. We achieve this goal by proposing an approach to **incrementally maintain VisRec ranking**

Figure 1.3: Example of Lux, a VisRec system, operating in a Jupyter Notebook Environment. The “college” dataset is rapidly ingested into a dataframe and then profiled and visualized with Lux in an eager fashion and in quick succession, with little to no user code.

scores across data updates for a wide variety of common VisRec visualizations/actions and data updates in order to avoid costly recomputation.

Our contributions are as follows:

- We **define a set of ranking scores** commonly used in many VisRec systems that can all be decomposed into aggregates, which in turn can be efficiently maintained with respect to data updates, and cover every data-based analytical action in Lee et al.’s taxonomy. (Chapter 3)
- **We decompose and consolidate these ranking scores into a set of five core aggregates**. We also outline challenges and limitations posed by maintaining these scores and associated aggregates. (Chapter 3)
- We **propose a set of five core data operators that cover common data updates across tabular data**, as well as provide methods to maintain the five core aggregates with respect to these operators. **We present a cost model** for estimating whether to incrementally maintain ranking score aggregates or recompute ranking scores from scratch upon a batch of updates. (Chapter 4)
- We implement our system in Lux and evaluate our system’s performance, namely time taken to maintain ranking scores during updates. We show our system scales well with dataset size, as well as demonstrate the efficacy of our cost model by showing our system is **as fast or faster than naive full recomputation of ranking scores** for datasets with many ranking scores. (Chapter 5)

1.4 Related Work

Early VisRec systems [19, 18, 28] used user-identified data fields, coupled with perceptual rules-of-thumb, to generate and recommend visual encodings for visualizations. Further mixed-initiative systems [25, 33, 32, 29, 31, 13, 15, 11] provided an enhanced set of user-facing capabilities such as high-level query languages or visualization specifications for a space of potential visualizations, coupled with *data-based recommendations*. Our work focuses on covering common statistics for data-based recommendation systems: systems that recommend visualizations based on data composition or statistical properties, otherwise known as *bottom-up data-driven inquiries* [16]. Prior work by Graefe et al. classifies a set of sufficient summary statistics for relational databases, as well as methods for efficiently computing them [8]. However, our work focuses on VisRec systems specifically, as well as maintaining statistics with respect to updates.

To effectively capture a wide range of common statistics for data-based recommendations, we analyze and decompose the VisRec system analytical action categories created by Lee et al.'s VisRec taxonomy [14]. Other taxonomies for visual analytics recommendations (not necessarily VisRec systems specifically) exist: Kaur et al. [12] outline statistical "data-characteristics oriented" strategies directly employed by the same VisRec systems already covered by Lee et al.'s taxonomy, such as [13, 31, 29, 11, 33, 32, 15, 13]. Seo et al. [23] present a statistical framework for methods ranking visualizations within high-dimensional datasets: the framework specifies similar methods of ranking visualizations to Lee et al.'s taxonomy. For example, histograms/bar charts are ranked through calculating normality and uniformity of distribution, and scatterplots via Spearman's correlation, which match Lee et al.'s taxonomy. And Lee et al.'s taxonomy covers all VisRec categories and visualization types that Seo et al.'s framework outlines. However, no prior work we are aware of exhaustively consolidates a list of common VisRec system ranking scores and methods into a set of aggregates to maintain during updates.

Prior work has attempted to speed up general EDA workloads in a computational notebook environment via statistic collection [24] and lazy evaluation [34]. In addition, methods for efficient statistic computation and maintenance in tabular data exist, such as partition recomputation [26, 35]. However, no prior work we are aware of focuses on incremental *maintenance* of statistics/metadata in a VisRec context specifically, or across data-specific updates in iterative EDA workflows. Lux, a VisRec system [15, 17], computes statistics and ranking scores lazily (upon user invocation) and caches them, but invalidates its entire cache upon any updates. Our work, in contrast, focuses on *maintenance* and not collection, through a variety of online incremental algorithms and efficient data structures.

Chapter 2

Overview of Approach

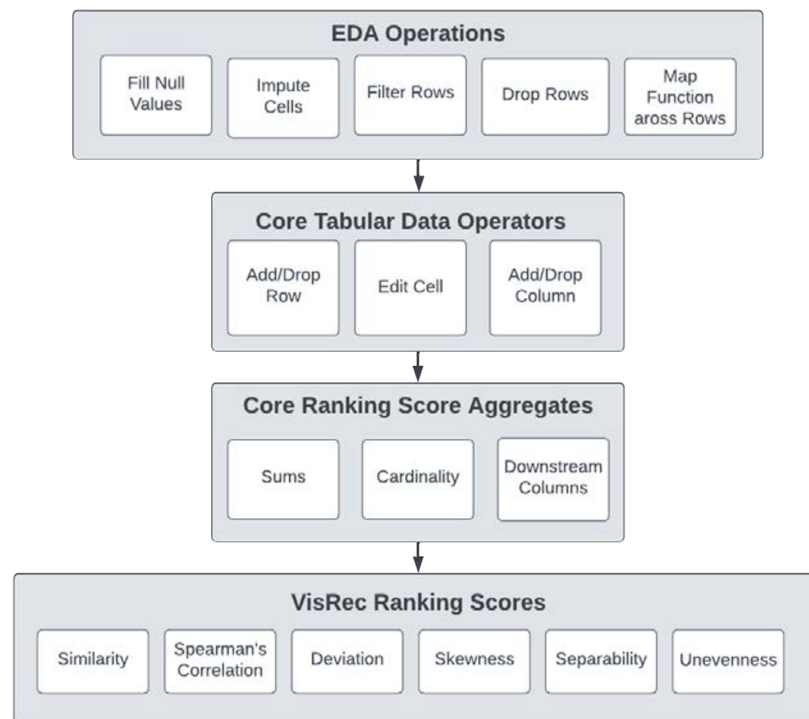

Figure 2.1: Illustrating our proposed approach for incrementally maintaining ranking scores. Common EDA operations can be composed of core tabular data operators, each of which efficiently update a set of core ranking score aggregates. These aggregates are used to efficiently derive common VisRec ranking scores.

2.1 Defining Ranking Score Aggregates and Core Operators

As mentioned previously, our challenge is to cover a range of VisRec systems and ranking scores, as well as a range of data updates. To address the first issue, we use Lee et al.'s VisRec taxonomy to map and understand the design space of common visualization and analytical actions in VisRec systems, as well as choose a feasible set of ranking scores that covers all categories within the taxonomy. In Chapter 3, we define each ranking score choice, as well as decompose each score into their constituent aggregates (e.g., cardinality computation or sums). We see that we are able to **decompose the range of scores into five core aggregates** that need to be maintained.

The second part of our challenge stems from the wide range of possible data updates during EDA. We constrain our data to tabular formats, and present a set of **core tabular data operators (primitives)** that can be composed to make up various data updates in Chapter 4. We also show how each of these core operators efficiently update the five core aggregates mentioned earlier.

2.2 Efficiently Maintaining Ranking Scores

Our goal is to incrementally maintain ranking scores with respect to common EDA data updates. These EDA data updates can be composed of our core tabular data operators, each of which, in turn, incrementally update our core ranking score aggregates. These aggregates can be used to efficiently derive VisRec ranking scores. Figure 2.1 illustrates this flow. We also present a cost model to estimate when it is computationally cheaper to incrementally update aggregates or to recompute ranking scores/statistics from scratch, thus maintaining ranking scores as fast as or faster than full ranking score recomputation from scratch.

Chapter 3

VisRec System Ranking Scores

Computing ranking scores for VisRec Systems is costly. We summarize a set of ranking scores that covers many visualization and analytical action types, including all categories listed in Lee et al.'s [14] analytical action categories, and decompose them into a set of core aggregates, to significantly reduce the effort needed to maintain the entire set of scores. We only need to maintain the set of core aggregates with respect to updates, to maintain the entire set of scores.

While Lee et al. does not exhaustively enumerate all possible analytical types, they map the design space of common recommendation categories for operational and characteristic analytical actions in many VisRec systems [29, 25, 32, 31, 15, 33, 11, 13, 19, 13, 31] as well as other visual analytics taxonomies [23, 22, 12]. We also address the taxonomy's limitations in the "Challenges and Limitations" section of this chapter.

3.1 VisRec System Taxonomy Overview

3.1.1 Analytical Actions Overview

Lee et al.'s taxonomy (Figure 1.1) outlines ten major analytical action categories drawing from Online Analytical Processing (OLAP) [9], separated into two major action categories: *operational* and *characteristic* actions. *Operational* actions are invoked by the user (such as adding a filter or additional category/data variable to the current view), while *characteristic* actions reveal underlying data qualities such as distribution or correlation. Actions are further grouped into *context-dependent* if they depend on a current view (a currently displayed visualization), or *context-independent* otherwise.

Operational actions must be context dependent, as they apply data-oriented operations on the currently displayed visualization. The operational action categories are as follows:

- **Enhance**: add an additional dimension or attribute to the current view. For example, if a user selects attribute X, enhance may show attributes X and Y.

- **Filter (add and swap)**: Add or swap filter F onto the current selection. If a user selects attributes X and Y, adding a new filter F will display visualizations with attributes X and Y, and filter F applied. Users are able to swap filter F with another filter F'.

- **Generalize (attribute)**: remove a dimension or attribute from the current view. If a user selects attributes X and Y, generalize may show visualizations with only attribute X.

- **Generalize (value)**: remove a filter from the current view. If a user selects attributes X and Y with filter F, generalize will display visualizations involving X and Y, without F.

- **Pivot**: show possible visualizations if one attribute within the current view is replaced by another attribute. If a user selects attributes X and Y, pivot will display visualizations involving X and another attribute Y', or another attribute X' and Y.

Meanwhile, characteristic actions reveal visual and statistical characteristics of data; they can be both context-dependent and context-independent. The characteristic action categories are listed below:

- **Correlation**: highlight relationships between two quantitative fields or columns via scatterplots

- **Distribution**: display univariate distributions within the dataset; quantitative fields may be displayed as a histogram, while categorical/qualitative data may be displayed as a bar chart aggregating over a categorical/qualitative field, with the default measure being $COUNT$.

- **Similarity/Difference**: display data patterns or characteristics that are visually similar/different from the current view.

3.1.2 Ranking Scores Overview

VisRec systems are able to generate and search through many visualizations, with some visualizations being more valuable to end-users than others. In many VisRec Systems, visualizations are ranked by a ranking score, or "interestingness objective" [14, 29, 15, 31, 33, 32, 19, 13, 11]. Ranking visualizations by score increases the utility of displayed visualizations for end users, and also allows VisRec systems to prune computation (only materialize the top k highest ranking visualizations) [15, 29]. We consider commonly occurring visualization types in VisRec systems, such as histograms, bar charts, line charts, colored and uncolored scatterplots. Visualizations can be ranked based on visualization type or user intent: users may be more interested in histograms that are more skewed or scatterplots with high degrees of correlation, or in visualizations similar to the current visualization displayed.

Within Lee et al.'s taxonomy (and others such as [23, 12]), visualizations within the characteristic analytical action categories are ranked separately within each action category: visualizations in the **Distribution** category are commonly ranked based on skewness, while **Correlation** visualizations are ranked based on monotonicity between two quantitative variables. Visualizations within the **Similarity/Difference** actions are ranked based on deviation (via distance metric such as Euclidean distance) from the current visualization displayed. These ranking metrics aim to represent important or noticeable characteristics expressed by visualizations within the characteristic action categories.

Meanwhile, Lee et al. state that ranking scores for operational analytical actions are based on data characteristics *determined by the target visualization type (e.g. bar chart, histogram)* of the recommended visualizations [14], intended to identify visualizations that are unexpected or particularly insightful for their category. As an example, users may be more interested in histograms that are more skewed, or scatterplots that display high degrees of correlation. Histograms and bar/line charts without a filter are ranked by unevenness, or non-uniformity, while histograms and bar/line charts with a filter are ranked based on deviation between filtered and unfiltered views. Uncolored scatterplots are ranked by mutual information (e.g., Spearman's correlation), while colored scatterplots are ranked by class or category separability [4].

3.2 Defining Existing Ranking Scores and Aggregates

We define and justify mathematical ranking metrics for each analytical action category in Lee et al.'s taxonomy in the following section. We also present necessary constituents (aggregates) for ranking scores, and synthesize these aggregates into a minimal set to maintain with respect to updates. Again, Lee et al.'s taxonomy and these ranking scores do not cover a comprehensive set of all possible visualization and analytical action types, but rather synthesize common visualization and analytical action categories.

3.2.1 Characteristic Actions

As mentioned in the previous section, characteristic action visualizations are ranked within each individual action category, and are designed to capture statistical visual characteristics such as correlation or skew. We define a ranking score and its associated aggregates for each characteristic action in the following section.

3.2.1.1 Correlation

Correlation between two quantitative columns can be ranked by **monotonicity** or similar measures, as seen in [15, 23, 31, 11, 19, 29]. Monotonicity can be measured by the square of the **Spearman Correlation Coefficient**. We choose Spearman over other correlation measures such as Pearson and Kendall due not only to its ability to capture monotonic relationships, but also its ability to be incrementally computed and maintained. The Spearman

Correlation Coefficient can be calculated between two quantitative columns X and Y of length n (where X_i and Y_i denote a single element at index i):

$$\text{Spearman}(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\overline{Y})^2}}$$

where $\overline{X} = \frac{1}{n}\sum_i^n X_i$, and $\overline{Y} = \frac{1}{n}\sum_i^n Y_i$.

We can rewrite the formula into a one-pass formula using terms derived from Welford's online variance algorithm [6] and the original formula:

$$\text{Spearman}(X,Y) = \frac{n\sum_{i=1}^{n} X_iY_i - \sum_{i=1}^{n} X_i \sum_{i=1}^{n} Y_i}{\sqrt{n\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2}\sqrt{n\sum_{i=1}^{n} Y_i^2 - (\sum_{i=1}^{n} Y_i)^2}} \tag{3.1}$$

$$= \frac{(\sum_{i=1}^{n} X_iY_i) - n\overline{XY}}{\sqrt{(\sum_{i=1}^{n} X_i^2) - n\overline{X}^2}\sqrt{(\sum_{i=1}^{n} Y_i^2) - n\overline{Y}^2}} \tag{3.2}$$

Let $\hat{X} = \sum_{i=1}^{n} X_i$, $\hat{Y} = \sum_{i=1}^{n} Y_i$, $a = \sum_{i=1}^{n} X_i^2$, $b = \sum_{i=1}^{n} Y_i^2$, $c = \sum_{i=1}^{n} X_iY_i$. These will be the aggregates we accumulate to compute Spearman(X,Y) efficiently. We see we can rewrite Spearman(X,Y) from Equation 3.2 as:

$$\text{Spearman}(X,Y) = \frac{(\sum_{i=1}^{n} X_iY_i) - n\overline{XY}}{\sqrt{(\sum_{i=1}^{n} X_i^2) - n\overline{X}^2}\sqrt{(\sum_{i=1}^{n} Y_i^2) - n\overline{Y}^2}} \tag{3.3}$$

$$= \frac{c - \frac{\hat{X}\hat{Y}}{n}}{\sqrt{a - \frac{\hat{X}^2}{n}}\sqrt{b - \frac{\hat{Y}^2}{n}}} \tag{3.4}$$

We see that to compute Spearman(X,Y), and maintain it in constant time with respect to updates to X and Y, **we can accumulate n and the following five terms**: $\sum_i X_i$, $\sum_i Y_i$, $\sum_i X_i^2$, $\sum_i Y_i^2$, $\sum_i X_iY_i$. These are the same as $\hat{X}, \hat{Y}, a, b$, and c from Equation 3.4.

3.2.1.2 Distribution (Quantitative)

Visualizations within the distribution category for quantitative data variables are typically represented as histograms [15, 29, 13, 23, 11]. A quantitative column X of length n's histogram is commonly ranked against other histograms in the same category based on **skewness**, defined as $\frac{\overline{X}^3}{\sigma_X^3}$, where $\overline{X}$ denotes mean and σ_X^2 denotes variance. Much work already exists on computing and storing mean and variance in an online fashion [6]. We can derive σ_X^3 from σ_X^2 through exponentiation. Recall that we calculate mean and variance as follows:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{3.5}$$

$$\sigma_X^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n} = \frac{\sum_{i=1}^{n} X_i^2}{n} - \overline{X}^2 \tag{3.6}$$

Let $a = \sum_i X_i$, $b = \sum_i X_i^2$. We can rewrite mean and variance as:

$$\overline{X} = \frac{a}{n} \tag{3.7}$$

$$\sigma_X^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n} = \frac{b}{n} - (\frac{a}{n})^2 \tag{3.8}$$

We see that we can **accumulate and maintain** n**,** $\sum_i X_i$ **(i.e.,** a**), and** $\sum_i X_i^2$ **(i.e.,** b**)**, where X_i denotes a single element of X at index i, to maintain mean and variance, and therefore skewness.

3.2.1.3 Distribution (Qualitative/Categorical)

Visualizations within the distribution action for non-quantitative (categorical) data are typically displayed as bar charts, with a chosen aggregate such as *COUNT* or *AVG* over the categorical field [15, 17, 23, 29, 32, 11]. For example, in a dataset with countries as categorical data, users may be interested in a bar chart displaying average population per country, or even the number (count) of each country represented in the data. These bar charts can be ranked by **unevenness**, or non-uniformity. Many measures of non-uniformity exist, but one of the simplest that follows our theme of maintaining column sums (see Correlation, Distribution actions) is Euclidean distance between normalized bar chart values and a flat distribution.

Formally, for categorical column X, we can construct aggregate values vector V by aggregating over X, and measure $L_2(V, V_{\text{flat}})$, where V_{flat} denotes the flat uniform distribution based on $|V|$, where $|V|$ denotes cardinality of V. In other words, each entry of V_{flat} is $\frac{1}{|V|}$. Note that V can be normalized by dividing each value by $\sum V_i$ upon calculation. Cardinality can be optionally used to weight ranking scores further, as done in VisRec systems such as Lux [15]. Let $X = V$, $Y = V_{\text{flat}}$, both of length n. X_i denotes the i'th entry of X. Recall the Euclidean distance formula:

$$L_2(X, Y) = \textstyle\sum_{i=1}^{n}(X_i - Y_i)^2 = \sum_{i=1}^{n}(X_i^2 - 2 \cdot X_iY_i + Y_i^2)$$

Recall that $Y = V_{\text{flat}}$, which means $Y_i = \frac{1}{|V|}$ for all i. Then, the formula becomes

$$L_2(X, Y) = L_2(V, V_{\text{flat}}) = \textstyle\sum_{i=1}^{n} V_i^2 - \frac{2}{|V|}\sum_{i=1}^{n} V_i + \frac{n}{|V|^2}$$

As before, let $a = \sum_{i=1}^{n} V_i$, $b = \sum_{i=1}^{n} V_i^2$, and $c = |V|$. We can rewrite $L_2(V, V_{\text{flat}})$ as:

$$L_2(V, V_{\text{flat}}) = b - \frac{2}{c}a + \frac{n}{c^2}$$

Therefore, upon updates to X, to maintain $L_2(V, V_{\text{flat}})$, **we must maintain V's aggregate values (i.e., V itself), as well as $\sum_{i=1}^{n} V_i$, $\sum_{i=1}^{n} V_i^2$, and** $|V|$ (i.e., a, b, and c in the example above).

3.2.1.4 Similarity/Difference

When considering a Similarity/Difference analytical action, VisRec systems tend to measure similarity/difference between a given distribution and a candidate distribution. Similarity/difference between two distributions can be measured using a variety of techniques: Earth Mover's Distance, Euclidean Distance, Kullback-Leibler Divergence (K-L divergence), and Jenson-Shannon Distance are a few of the many that have been explored in VisRec systems [29, 15, 33, 32, 11]. For simplicity, and to further our strategy of maintaining forms of data sums, we elect to use Euclidean Distance. Formally, for a candidate distribution of values Y constructed via the *Distribution* action, and given distribution (i.e. current view) X, we can measure similarity/difference via $L_2(X, Y)$ between measure values of each distribution. Note that some data interpolation may be necessary to fix the number of data points to a constant when computing $L_2(X, Y)$; interpolation is out of scope for this work, as it incurs additional complexity that makes it difficult to maintain scores.

Recall from the previous section the formula for $L_2(X, Y)$:

$$L_2(X, Y) = \sum_{i=1}^{n} (X_i - Y_i)^2 \tag{3.9}$$

$$= \sum_{i=1}^{n} (X_i^2 - 2 \cdot X_i Y_i + Y_i^2) \tag{3.10}$$

$$= \sum_{i=1}^{n} X_i^2 - 2 \sum_{i=1}^{n} X_i Y_i + \sum_{i=1}^{n} Y_i^2 \tag{3.11}$$

As before, let $a = \sum_i X_i^2$, $b = \sum_i X_i Y_i$, and $c = \sum_i Y_i^2$. We can rewrite Equation 3.11 as:

$$L_2(X, Y) = a - 2b + c$$

To maintain $L_2(X, Y)$, **we must maintain candidate distribution values Y (i.e., Y itself), as well as $\sum_i X_i^2$, $\sum_i X_i Y_i$, and $\sum_i Y_i^2$.**

3.2.2 Analytical Actions

Analytical action visualizations are typically ranked based on visualization type (e.g., colored scatterplot, bar chart); as mentioned in previous sections, ranking scores are designed to reveal unexpected or insightful visual characteristics.

3.2.2.1 Bar Charts, Histograms without Filter

Bar charts without a filter are described in the "Distribution (Qualitative/Categorical)" Action category above. Histograms without a filter are described in "Distribution (Quantitative)".

3.2.2.2 Bar Charts, Histograms with Filter

Bar Charts and Histograms with a filter, however, are ranked based on deviation (distance) between filtered and unfiltered values [14, 15, 25]. For example, a user may choose to filter a distribution by a specific country (e.g., given average SAT scores, filter scores for the United States only). Again, many distance metrics such as Earth Mover's Distance and Jenson-Shannon Distance exist in VisRec Systems [29, 19], but we choose Euclidean distance for simplicity and to accentuate our strategy of maintaining forms of data sums.

Formally, for column X and filter F, we can rank filtered view $Y = X_F$ by computing $L_2(X, Y)$ between filtered and unfiltered values. We can define filtered view $Y = X_F$ as column X filled with 0 everywhere filter F is false, and the original values of X where filter F is true. Maintaining $L_2(X, Y)$ is achieved through **exactly the same strategy used for the Similarity/Difference** action above, except the candidate distribution is replaced by filtered values $Y = X_F$. To maintain $L_2(X, Y)$, **we must again maintain $Y = X_F$ (the filtered values vector) itself, $\sum_i X_i^2$, $\sum_i Y_i^2$, and $\sum_i Y_i X_i$**.

3.2.2.3 Uncolored Scatterplots

Uncolored Scatterplots are more valuable if they display a high degree of dependence between two measures, which can be measured by heuristics such as mutual information or Spearman's correlation [4, 23]. We choose to compute and maintain Spearman's correlation, as it is already in use to compute monotonicity for the Correlation action described in previous sections. Recall that **monotonicity is computed via the square of the Spearman's correlation**. The aggregates to maintain **are the same as the Correlation action**, where we already maintain Spearman's correlation.

3.2.2.4 Colored Scatterplots

Colored scatterplots are ranked based on separability, that is, if the colors for each category distinctly separate the data points on the scatterplot. There exists many different measures of separability [22, 7, 23, 4]. Depending on the separability measure, decomposing

and efficiently maintaining the necessary heuristic with respect to updates may be difficult, especially for complex multidimensional class separability measures. For the scope of this work, we elect to use the efficiently-maintainable measure of per-class mean and variance, to efficiently compute separability measures such as the Fisher Criterion [7]. For two classes x_1 and x_2, we can compute the Fisher Criterion between classes as $\frac{(\overline{x_1}-\overline{x_2})^2}{\sigma_1^2+\sigma_2^2}$, where $\overline{x_1}$ denotes mean of class x_1 and σ_1^2 denotes variance of class x_1. Then, to quickly compute the Fisher Criterion between any two classes, we can maintain mean and variance per class; maintaining mean and variance are already discussed in the Distribution (Quantitative) section.

3.3 Decomposing Ranking Scores

Ranking Score for Column X and additional Column Y	$\sum_i X_i$	$\sum_i Y_i$	$\sum_i X_i^2$	$\sum_i Y_i^2$	$\sum_i X_i \cdot Y_i$	$\lvert X \rvert$
Correlation: Spearman$(X, Y)^2$	✓	✓[1]	✓	✓	✓	
Skewness: μ_X^3/σ_X^3	✓		✓			
Monotonicity: Spearman(X, Y)	✓	✓	✓	✓	✓	
Separability: Class Mean/Variance	✓		✓			
Similarity: $L_2(X, Y)$, $Y =$ Given Distribution (Current View)			✓	✓	✓	
Deviation: $L_2(X, Y)$, $Y =$ Filter F applied on X (known as X_F)			✓	✓[2]	✓	
Unevenness: $L_2(X, Y)$, $X = \gamma_X$ (or V_X), $Y = V_{\text{flat}}$	✓		✓			✓

Table 3.1: Decomposing ranking scores for Data Variable (Column) X and (optional) external column Y into their respective aggregates. The columns of the table are listed as follows: sum of X, sum of Y, sum of X's squared elements, sum of Y's squared elements, inner product of X and Y, cardinality of X. $\gamma_X = V_X$ is values over an aggregate over X, X_F represents filtered values, and V_{flat} represents the flat uniform distribution based on cardinality.

Table 3.1 shows the ranking scores for column X decomposed into their respective aggregates. The columns of the table are listed as follows: sum of X, sum of Y, sum of X's squared elements, sum of Y's squared elements, inner product of X and Y, cardinality of X. $\gamma_X = V_X$ is values over an aggregate over X, X_F represents filtered values, and V_{flat} represents the flat uniform distribution based on cardinality. Note that correlation, skewness, and monotonicity are only defined for quantitative data variables, while similarity, deviation, and unevenness cover both qualitative (categorical) and quantitative data.

We see that to maintain ranking scores for Column X and additional Column Y, we can maintain the following core set of aggregates:

1. Downstream columns such as aggregate values $V_X = \gamma_X$ and filtered column X_F
2. $\sum_i X_i$
3. $\sum_i X_i^2$
4. $\sum_i X_i Y_i$ for external column Y

[1] Y represents another quantitative column
[2] Y represents filtered column X_F

5. $|X|$ (cardinality of X)

Table 3.2 presents these five core aggregates.

Note that additional column Y and downstream columns such as aggregate values V_X also have their own sets of aggregates to maintain from aggregates 2, 3, 4, and 5. For example, for filtered columns, we must maintain a sum of their elements squared and pairwise sum between filtered and unfiltered columns. For colored scatterplots, where separability is the ranking score criterion, each aggregate is maintained per class. Given that column pre-processing (filters, aggregate values computation) is complete, each of the ranking scores can be computed in *constant or* $O(1)$ time using these aggregates. **Our goal becomes to efficiently maintain these aggregates with respect to updates**, to maintain ranking scores.

Downstream columns such as aggregate values $V_X = \gamma_X$ and filtered column X_F	$\sum_i X_i$	$\sum_i X_i^2$	$\sum_i X_i Y_i$	$\|X\|$

Table 3.2: Aggregates to maintain for Column X and external Column Y.

3.4 Limitations

As discussed previously, maintaining distribution values aggregated over categorical data can be difficult depending on the method of aggregation. We discuss efficient maintenance of aggregated distribution values further in Chapter 3. In addition, a variety of other more-complex ranking scores exist for each visualization type, such as increasing complex separability measures [22]. Visualization types can be extremely complex. Our work seeks to cover common visualization types (e.g., 3D geographical charts or multi-dimension visualizations) and analytical actions found in VisRec systems (that can be efficiently maintained), and covering increasingly complex ranking heuristics and visualization types is left to future work.

Chapter 4

Maintaining VisRec Ranking Score Aggregates

Recall, our goal is to efficiently maintain VisRec ranking scores with respect to updates. Existing VisRec systems such as Lux [15] and AutoViz [3] compute statistics and ranking scores upon visualization, but recompute statistics and ranking scores from scratch upon any data updates, as shown in Figure 4.1. In the previous section, we enumerated five core aggregates to maintain with respect to updates in order to maintain our ranking scores. To better serve VisRec systems with a tabular data model specifically, we briefly discuss data-dependent operations (that modify ranking scores) for tabular data specifically, and present a set of primitive operations that cover tabular data updates (despite the sheer number of operators in modern tabular data formats such as dataframes [21]). We map our primitive operations to efficient updates of the five core aggregates. We then propose a cost model for determining when to incrementally update the ranking score aggregates (and thus maintain ranking scores for visualizations) or to recompute ranking scores entirely upon a batch of data updates.

4.1 Tabular Data Operations and Maintaining VisRec Scores

4.1.1 Data and Metadata Operations

Data tables are organized into rows and columns (with a "cell" being a row, column entry) as seen in Figure 4.2, and contain a wide breadth of operations for both data (such as relational-style joins, projection, selection) and metadata (modifying column labels, indices, row labels, etc.) [21, 24]. Columns can span non-numeric and numeric types, as well as predefined, lazily-induced, or dynamic types. For the scope of this work, we consider data-dependent operators that modify data and not metadata such as column type or row label. We leave direct analysis of large-scale transformations such as transpose or relational-style

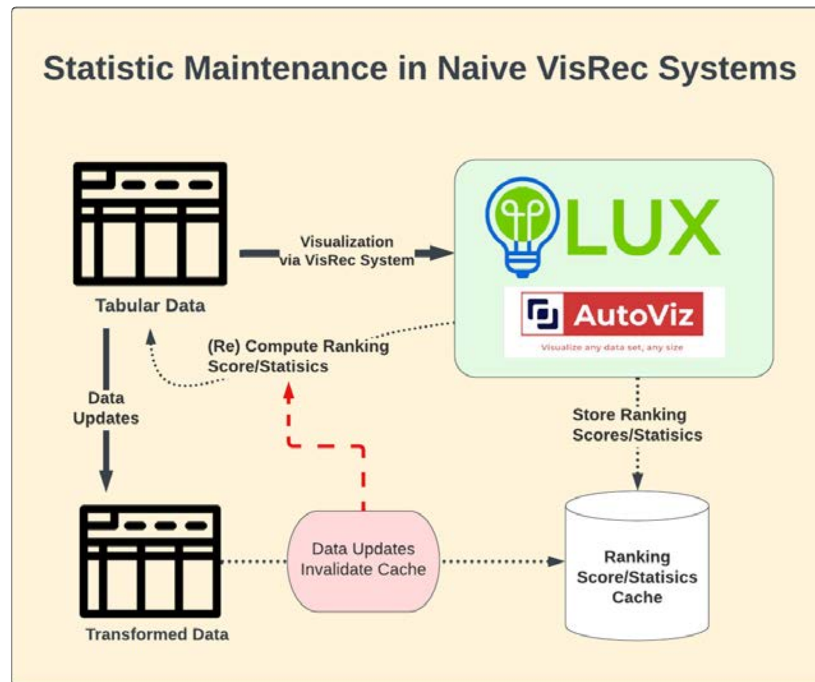

Figure 4.1: Existing VisRec systems such as Lux [15] and AutoViz [3] compute ranking scores upon visualization, but recompute ranking scores from scratch upon updates, as shown in Figure 4.1.

joins for future work; however, some of these transformations can be composed of the core operators described in the next section.

4.1.2 Core Primitive Tabular Data Operators for Maintaining VisRec Scores

We consider the following data operations that directly modify tabular data:

1. Adding a column
2. Removing a column
3. Adding a row
4. Removing a row
5. Editing a specific cell

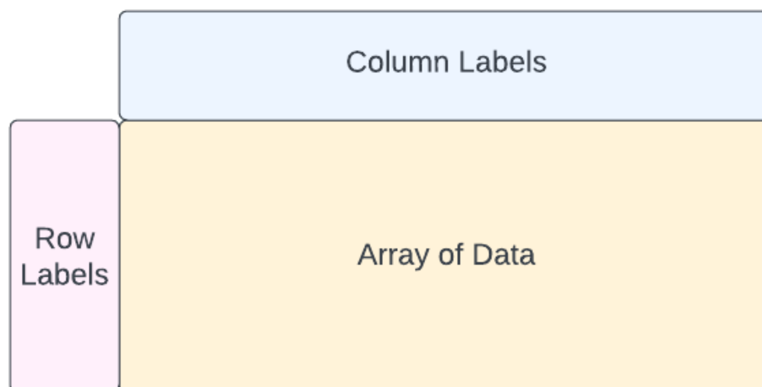

Figure 4.2: A dataframe, an example of a tabular data model, with row and column labels. A dataframe with N rows and M columns will have $N \cdot M$ cells in "Array of Data".

These core operations can be composed to capture data (not necessarily metadata) updates, and thus serve as our primitive units of operation for maintaining ranking scores. Many common EDA operations, such as cleaning data by dropping rows, filling or imputing missing values, deriving new columns, and exploding individual cells can be composed using these core tabular data operations.

We describe the process of maintaining ranking scores when applying these core operations in the following section.

4.1.3 Maintaining Aggregates with Respect to Updates

We proceed under the assumption that VisRec system statistics and visualizations are computed/generated per column for a tabular dataset, as seen in current systems such as Lux and AutoViz [15, 17, 3]. Recall from Chapter 2 that our five aggregates to compute and maintain for column X and external column Y consist of the following:

1. Downstream columns such as aggregate values $V_X = \gamma_X$ and filtered column X_F
2. $\sum_i X_i$
3. $\sum_i X_i^2$
4. $\sum_i X_i Y_i$ for external column Y
5. $|X|$ (cardinality of X)

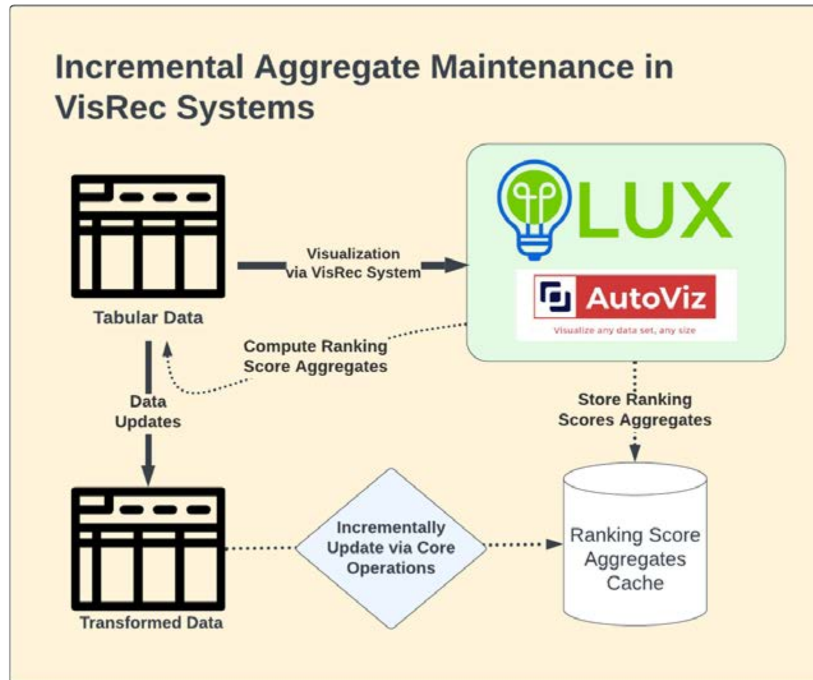

Figure 4.3: We incrementally maintain VisRec ranking score aggregates upon data updates, instead of fully recomputing statistics from scratch as seen in Figure 4.1.

Assuming statistics are generated per column (as done in VisRec systems such as [15, 3, 29, 13, 32]), adding a column requires complete computation of the five aggregates above (while computing and ranking possible visualizations and additional columns such as filters), while removing a column merely removes the five aggregates for the removed column entirely, as well as and associated downstream columns. Computing aggregates for an added column can be done lazily upon the next VisRec system invocation, while deleting a column incurs little overhead when deleting aggregates. However, **row operations require iterating over columns and applying an add or delete element operation to the aggregates for each column**. Editing any random cell involves deleting and then adding an element to the corresponding aggregates for the cell's column. Figure 4.3 illustrates our proposed scheme. We enumerate processes for adding elements to and removing elements from each aggregate in Table 3.2 below.

4.1.3.1 Maintaining Downstream Columns for Column X

We describe maintaining the first column of Table 3.2: downstream columns such as aggregate values $V_X = \gamma_X$ and filtered column X_F. For this work, it is assumed downstream

columns such as aggregate values V_x can be maintained efficiently; this is possible for boolean-filtered columns (each element addition/removal passes through an additional filter check) and common aggregate values such as *COUNT* and *AVERAGE*. *COUNT* can be efficiently maintained with respect to element additions and deletes by mapping each distinct column category/value to its count via an in-memory hash table, while *AVERAGE* can be maintained by maintaining a sum and count for each column category/value. However, this is harder for aggregate values such as *MIN* and *MAX* due to memory or runtime constraints. Maintaining these harder aggregate values is out of scope for this work.

4.1.3.2 Maintaining Sums for Column X

To maintain the second, third, and fourth columns of Table 3.2, we must accumulate and maintain sums in memory. These sums are $\sum_i X_i$, $\sum_i X_i^2$, and $\sum_i X_i Y_i$ for external column Y. Element removal involves subtracting the removed element from each sum, while element additions require adding the new element to each sum. Upon updates to column X, the fourth column of Table 3.2 (pairwise sum $\sum_i X_i Y_i$ for external column Y) can be updated by querying row information from column Y (adding a row means adding an element from both columns X and Y to the aggregate, deleting a cell on row j means subtracting $X_j \cdot Y_j$ from the aggregate, where X_j denotes the j'th element of column X), and applying the corresponding updates to $\sum_i X_i Y_i$ accordingly.

4.1.3.3 Maintaining Cardinality

We maintain the fifth column of Table 3.2, cardinality or $|X|$, using existing techniques. Much previous work has been done on maintaining cardinality information in a variety of data systems [10]. In many cases, cardinality can be maintained through by mapping unique elements to their value counts (same as maintaining the *COUNT* value aggregate in the previous section) as done in [15], or through data structures such as a bloom histogram in memory-intensive use cases at scale. Element removal involves decrementing the element's value count (and removing the element when its value count is zero), while element addition involves incrementing an existing value count or inserting a new unique value with a single value count. Cardinality is derived from the number of unique values (keys in the hash table or bloom histogram).

4.2 Cost Model for Updates

As mentioned earlier, large-scale data transformations such as adding many rows can be expressed in terms of our core tabular data operations. We present a cost model to, upon updates, determine whether to recompute aggregates entirely from scratch or incrementally update existing aggregates in a step-by-step fashion by applying the individual data operators and their corresponding aggregate modifications mentioned earlier. Our cost model does not consider system factors such as memory limits and CPU/cores and contains little query

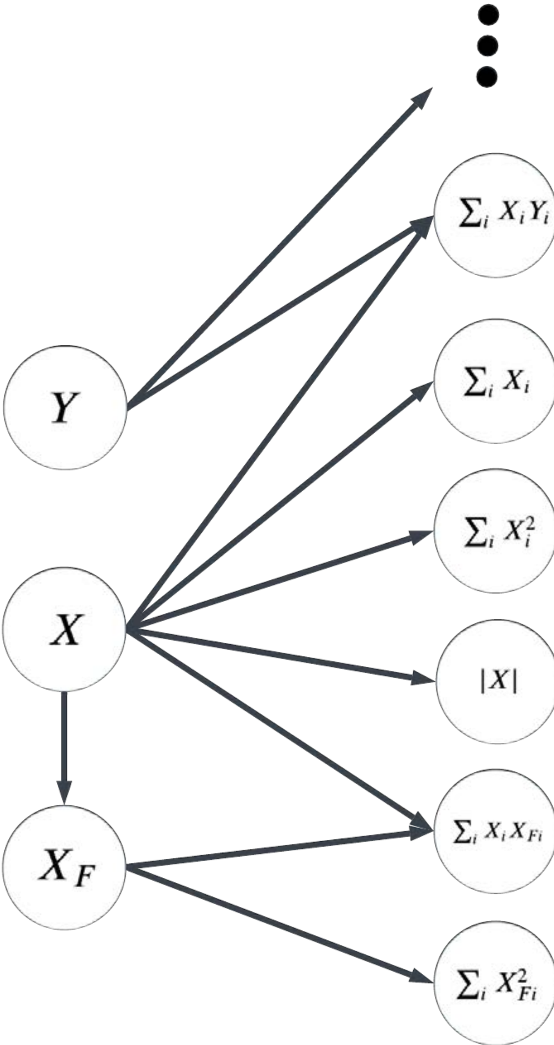

Figure 4.4: A representation graph displaying quantitative column X. X is modeled as a source, with directed edges to its aggregates $\sum_i X_i$, $\sum_i X_i^2$, $|X|$, and $\sum_i X_i Y_i$, as well as its downstream, filtered column X_F. Note that Y, an additional column, and X_F also have their own dependencies (aggregates to maintain), modeled as additional connected nodes.

planning/estimation factors. Improving the cost model is an area for future research work that we address in the Limitations section.

4.2.1 Modeling Column Relationships and Aggregates

To model cost and scope of required updates when updating a column, we model our stored aggregates using a graph. Edges in the graph will represent necessary updates and serve as the basis for our cost model. Each vertex or node represents a column or aggregate.

We construct a directed-acyclic graph (DAG), with dataset columns represented as sources and associated aggregates as sinks. Directed edges capture dependencies and are drawn between column and associated aggregates or downstream columns. Downstream columns also have associated aggregates to maintain. For instance, to model quantitative column X with filter F (and thus downstream column X_F, represented a filtered version of X), we create sink nodes for $\sum_i X_i$, $\sum_i X_i^2$, $|X|$, and $\sum_i X_i Y_i$, where Y is an external column, and create a directed edge between X and these four nodes. Note that Y also has an edge to the node representing $\sum_i X_i Y_i$. We also create an edge between X and a node representing downstream column X_F, as well as create edges between X_F and its associated aggregates. Figure 4.4 illustrates the graph model capturing column-aggregate relationships.

4.2.2 Incremental Updates versus Full Recomputation

In our graph representation, each edge represents a necessary update whenever its parent receives an update. Thus, the cost for updating the aggregates associated with column X a single time is the number of edges reachable from column X's source node (and can be derived via traversal). Denoting the number of edges reachable from column X as E_X, then the cost for k updates to column X is $k \cdot E_X$. Removing a column incurs no cost in our model, since it requires the minimal overhead of merely removing a source and its reachable descendant nodes. Adding a column requires determining necessary aggregates/downstream derived columns, and then adding the necessary source and dependency nodes, as well as the cost for computing the aggregates/dependencies for that column.

Then, because adding or removing a row incurs an update to every column in the data, the cost for adding or removing a row becomes the number of edges in the graph (due to the fact that each column and thus source is updated), which can be denoted as E. Editing a cell in column X, as described earlier, incurs cost E_X.

Given a tabular dataset D with N rows, the cost to recompute all ranking score statistics/aggregates from scratch can be roughly similar in computation to adding N individual rows, and thus can be modeled as $N \cdot E$ (number of rows multiplied by total number of dependency edges). And the cost for computing statistics/aggregates for a single column X, made up of N rows, can be modeled $E_X \cdot N$. Thus, for any given batch of row/column/cell updates (e.g. adding many rows, editing cells, removing a column), **if the cost of all applying all updates in the batch incrementally exceeds the cost of full recomputation** ($N \cdot E$), **we apply all updates and then compute aggregates/statistics from scratch.**

Otherwise, we can apply our updates and maintain our aggregates/statistics incrementally (step-by-step) using strategies we outlined in "Maintaining Aggregates with Respect to Updates."

4.3 Limitations

4.3.1 Tabular Data Operations

Our core operators (adding/removing columns, adding/removing rows, editing cells) do not explicitly cover large scale transformations such as group by aggregation or joins. While relational databases contain much work on query estimation and planning, other popular EDA tools such as dataframes in computational notebooks have a wide, expressive API. Much work is to be done in the areas of dataframe and VisRec-specific query estimation and optimization [21, 34, 24], as well as statistic collection. And our five core data-modifying operators do not cover metadata transformations such as transposing tables. These are directions for future work.

4.3.2 Cost Model Limitations

Our cost model does not consider system parameters such as cache size, memory, and CPU factors. It also assumes that updating downstream derived columns such as aggregated values over a categorical column or filtered columns is relatively similar to updating aggregates such as sums, and treats these updates as having the same cost. The model only loosely considers the number of rows or columns in the dataset, which could be a crucial factor in the computation time for ranking scores; more columns in the dataset means more ranking scores to maintain, whereas more rows means each individual ranking score takes longer to compute. Future work could weight edges by cost accordingly, or develop an even more fine-grained cost model that accounts for more complexity in collecting and updating dataset statistics such as system factors or file format. In addition, our cost model does not consider the shape of the dataset. Datasets with many columns have many possible visualizations and thus many ranking scores to maintain. Datasets with fewer columns have fewer ranking scores, and full recomputation may be faster than incremental update maintenance for these datasets.

Chapter 5

Implementation and Performance Evaluation

We implement our proposed method for maintaining VisRec system ranking scores in Lux, a VisRec system for pandas dataframes. Then we discuss use case examples of how common data science operations such as cleaning data can be composed of our core tabular data operations, and why they are inefficient in Lux. We show how the number of columns and number of rows affects computation time for performing our core operations and incremental updates. We then evaluate the performance of our strategy by measuring total wall-clock time (user-facing latency) for maintaining ranking scores against naively recomputing statistics/ranking scores from scratch, and also demonstrate the efficacy of our cost model.

5.1 Implementation in Lux

Lux is an open-source VisRec system with 4.5k GitHub stars as of March 2023 [17, 15]. Lux's core computation involves collecting statistics and metadata per dataframe column, then generating possible visualizations and computing ranking scores for each possible visualization, then materializing each visualization up to a specified number of visualizations (to enable early pruning of visualization generation). It caches these ranked visualizations, along with their respective ranking scores, but **invalidates its entire cache upon any data-modification** to the dataframe and recomputes all visualizations/ranking scores from scratch.

Lux explicitly implements **all the analytical actions and their respective ranking scores** in Lee et al.'s VisRec taxonomy [14], as well as various other specific scores such a Chi-square test for independence. We implement storing the five core aggregates (three sums, cardinality, and downstream derived columns such as filtered columns) per column in Lux, as well as the core tabular data operators for adding and deleting a row and as editing a cell, along with their respective updates to the five aggregates accordingly. Deleting columns and adding columns are built-in to Lux dataframes, and upon column addition/deletion the

five core aggregates are computed/removed accordingly. These specifications are exactly as described in Chapter 3.

5.2 Dataframe Use Case Examples

Our evaluation benchmarks consist of testing our core operators from Chapter 4, specifically adding and deleting rows, and editing cells. These three operations can compose many common EDA data updates such as imputing missing data (editing many cells), dropping missing data (dropping rows), filtering rows (dropping rows), and transforming rows (dropping then adding rows or editing cells).

5.3 Evaluation

5.3.1 Datasets and Experiment Setup

We evaluate our implementation on an AWS EC2 t3.xlarge instance with 4 cores and 16 GB of main memory, which is similar to modern laptops that EDA workloads in computational notebooks are run on. We run Python 3.7.16 with Lux-API version 0.5.1, and use the Python *time* library to performance benchmark. We measure user-facing latency (wall-clock time) taken to compute ranking scores in Lux.

We test how our implementation scales with respect to datasets with many rows and datasets with many columns. We use the Airbnb dataset [20] with 16 columns and about 50000 rows, and duplicate rows up to 2 million. We also use the UCI Crime and Communities dataset [2], with up to 128 columns and about 2000 rows. These datasets cover over 95% of the datasets in the UCI repository [5] and were used by the original Lux authors to benchmark Lux's performance [15].

5.3.2 Scaling with respect to Rows

Using the Airbnb dataset, we apply a fixed number of 10000 operations (5000 row deletes and 5000 row additions from random rows sampled from the original dataset), while fixing the number of columns (16) and increasing the number of rows from 250,000 to 2 million. Note that adding and deleting a row is equivalent in our implementation to editing 16 (number of columns in the original dataset) individual cells, and the two can be interchanged in terms of computation cost. We measure the **overall time taken to maintain ranking scores**. We also separately measure the time necessary to recompute ranking scores from scratch for our baseline for comparison. Figure 5.1 demonstrates this.

We see that increasing the number of rows does not significantly increase the computation time of our fixed number of 10000 data operations. This is because each data operation (add or delete rows, or equivalently edit cell) applies updates to a fixed number of aggregates, because there are a fixed number of ranking scores per column. However, for smaller datasets,

Latency of Computing Ranking Scores for 10000 Incremental Data Operations versus Full Computation in AirBnb Data

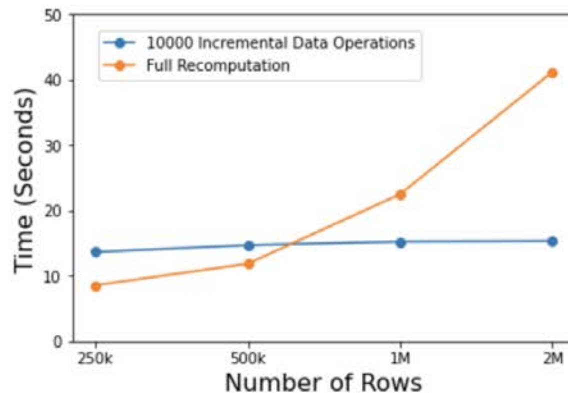

Figure 5.1: Within the Airbnb dataset, we apply a fixed number of 10000 operations (5000 row deletes and 5000 row additions), while fixing the number of columns (16) and increasing the number of rows from 250,000 to 2 million. Updating ranking scores takes a fixed amount of time regardless of number of rows. However, for smaller datasets, full recomputation is faster. We demonstrate in the next section that when the number of columns in a dataset increases (and thus the number of possible visualizations and ranking scores increases), our strategy becomes much more effective.

full recomputation is faster. This is because the AirBnb dataset does not have many columns, it does not have as many possible visualizations as datasets with many columns (and thus does not have as many ranking scores to compute/maintain). However, we demonstrate in the next section that when the number of columns in a dataset increases (and thus the number of possible visualizations and ranking scores increases), our strategy becomes much more effective.

5.3.3 Evaluation against Full Recomputation

Next, we evaluate the overall performance of our strategy on the UCI Crime and Communities dataset [2], with 128 columns and 2000 rows. We compare performance to Lux's existing method of recomputing statistics and ranking scores from scratch. We perform and benchmark a variable number of incrementalized data update operations, varying the number of row additions and deletes from 6.25% of the number of rows (125 rows) to 150% of the number of rows (3000 rows). Again, note that adding and deleting a row is equivalent in our implementation to editing 128 (number of columns) individual cells (i.e. adding a row is equivalent to editing 64 cells, or half the number of columns, because an edit involves both an

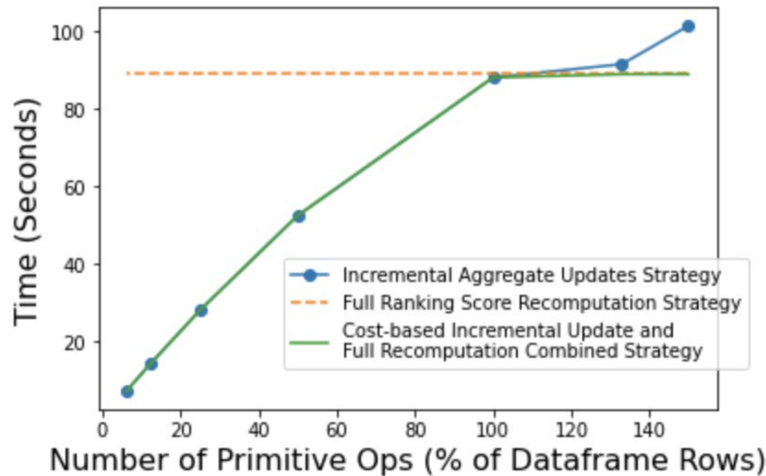

Figure 5.2: We compare efficacy of the incremental aggregate update strategy versus full statistic/ranking score recomputation in Lux on the Crime Data dataset (2000 rows, 128 columns), varying the number of row additions and deletes from 6.25% of the number of rows (125 rows) to 150% of the number of rows (3000 rows). We see that for row operations up to 100% of the number of rows (2000 rows), incrementally updating aggregates is preferred. For any more updates, full ranking score recomputation from scratch is faster, as demonstrated by our cost model.

element addition and deletion). We compare the latency of applying these incrementalized data operations and their respective aggregate updates to the process of full statistic/ranking score recomputation, shown in Figure 5.2. We see that for row operations up to 100% of the number of rows (2000 rows), incrementally updating aggregates is preferred. For any more updates, full ranking score recomputation from scratch is faster.

We see that the assumptions of our cost model are largely correct for datasets with many columns and thus large number of ranking scores: full recomputation is preferred when the number of row operations exceeds the number of rows in the dataset. In our cost model, recall that, given an aggregate dependency graph with E edges, k row updates costs $k \cdot E$, while full recomputation for a dataset with N rows costs $E \cdot N$. So in the case where $k > N$ (more row operations than 100% of number of dataset rows), full recomputation is preferred.

In short, for datasets with many columns and thus many ranking scores, **our cost-based approach is either as fast or faster than full recomputation when maintaining ranking scores under data updates**.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We propose a system for maintaining VisRec ranking scores under data updates in EDA workflows, for many common VisRec system analytical actions. We first survey ranking scores for a wide variety common of visualization types and analytical actions, and then provide a core set of five aggregates to compute and maintain per column, to compute the entire set of scores efficiently. We then provide five core tabular data operators (primitives) that can be composed to cover a wide variety of data modifications, and show how they update the five core aggregates incrementally and efficiently. We provide a cost model for updates, and present a strategy for determining when to recompute ranking scores from scratch or when to incrementally (step-by-step) update the five core aggregates for each operator. We implement our system in Lux, a popular open-source VisRec system and show how our system scales, as well as demonstrate the efficacy of our cost model by showing our system for maintaining ranking scores is as fast or faster than naive full recomputation of ranking scores under updates.

6.2 Future Work

VisRec systems are constantly evolving. Our approach characterizes a wide variety of common visualization types and analytical actions, and provides reasonable choices for ranking scores able to be maintained efficiently with respect to updates. However, new VisRec systems may implement analytical actions or visualization types, or ranking scores that are not covered by our system or are difficult to efficiently maintain.

In addition, as mentioned in [21, 34], the area of query planning and optimization for VisRec systems and EDA tools such as computational notebooks is being actively researched. Although our five core data update operators do compose to encompass many large-scale transformations such as joins, they do not necessarily do so in an optimized manner. Greater refinement on tabular data transformation coverage and optimization, as well as refining our

cost model based on system parameters, is grounds for future research. And techniques utilizing further resources such as multiple cores or background statistic collection during periods of computational notebook inactivity could provide further speedups, as explored in [34, 24].

Bibliography

[1] Kunal Agarwal. "Accelerating Visual Data Exploration via Sampling: A Case Study with Lux". In: *Technical Report No. UCB/EECS-2022-80* (2022). URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-80.html.

[2] *Analyzing UCI Crime and Communities Dataset*. https://www.kaggle.com/code/kkanda/analyzing-uci-crime-and-communities-dataset/data?select=crimedata.csv.

[3] AutoViML. *AutoViz*. https://github.com/AutoViML/AutoViz. Version 0.5.1. Oct. 26, 2022.

[4] Tuan Nhon Dang and Leland Wilkinson. "ScagExplorer: Exploring Scatterplots by Their Scagnostics". In: *2014 IEEE Pacific Visualization Symposium*. 2014, pp. 73–80. DOI: 10.1109/PacificVis.2014.42.

[5] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml. 2017. URL: http://archive.ics.uci.edu/ml.

[6] Andrey A. Efanov, Sergey A. Ivliev, and Alexey G. Shagraev. "Welford's algorithm for weighted statistics". In: *2021 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*. 2021, pp. 1–5. DOI: 10.1109/REEPE51337.2021.9387973.

[7] Keinosuke Fukunaga. "Chapter 5 - PARAMETER ESTIMATION". In: *Introduction to Statistical Pattern Recognition (Second Edition)*. Ed. by Keinosuke Fukunaga. Second Edition. Boston: Academic Press, 1990, pp. 181–253. ISBN: 978-0-08-047865-4. DOI: https://doi.org/10.1016/B978-0-08-047865-4.50011-9. URL: https://www.sciencedirect.com/science/article/pii/B9780080478654500119.

[8] Goetz Graefe, Usama M. Fayyad, and Surajit Chaudhuri. "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases". In: *Knowledge Discovery and Data Mining*. 1998.

[9] J. Gray et al. "Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS". In: *Proceedings of the Twelfth International Conference on Data Engineering*. 1996, pp. 152–159. DOI: 10.1109/ICDE.1996.492099.

[10] Hazar Harmouch and Felix Naumann. "Cardinality Estimation: An Experimental Survey". In: *Proc. VLDB Endow.* 11.4 (Dec. 2017), pp. 499–512. ISSN: 2150-8097. DOI: 10.1145/3186728.3164145. URL: https://doi.org/10.1145/3186728.3164145.

[11] Kevin Hu, Diana Orghian, and César Hidalgo. "DIVE: A Mixed-Initiative System Supporting Integrated Data Exploration Workflows". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. HILDA'18. Houston, TX, USA: Association for Computing Machinery, 2018. ISBN: 9781450358279. DOI: 10.1145/3209900.3209910. URL: https://doi.org/10.1145/3209900.3209910.

[12] Pawandeep Kaur and Michael Owonibi. "A Review on Visualization Recommendation Strategies". In: Feb. 2017. DOI: 10.5220/0006175002660273.

[13] Alicia Key et al. "VizDeck: Self-Organizing Dashboards for Visual Analytics". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD '12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 681–684. ISBN: 9781450312479. DOI: 10.1145/2213836.2213931. URL: https://doi.org/10.1145/2213836.2213931.

[14] D. Lee et al. "Deconstructing Categorization in Visualization Recommendation: A Taxonomy and Comparative Study". In: *IEEE Transactions on Visualization & Computer Graphics* 28.12 (Dec. 2022), pp. 4225–4239. ISSN: 1941-0506. DOI: 10.1109/TVCG.2021.3085751.

[15] Doris Jung-Lin Lee et al. "Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows". In: *Proceedings of the VLDB Endowment* 15.3 (2022), pp. 727–738. DOI: https://dl.acm.org/doi/10.14778/3494124.3494151.

[16] Doris Jung-Lin Lee et al. "You can't always sketch what you want: Understanding Sensemaking in Visual Query Systems". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (2020), pp. 1267–1277. DOI: 10.1109/TVCG.2019.2934666.

[17] lux-org. *Lux*. https://github.com/lux-org/lux. Version 0.5.1. Oct. 26, 2022.

[18] Jock Mackinlay. "Automating the Design of Graphical Presentations of Relational Information". In: *ACM Trans. Graph.* 5.2 (Apr. 1986), pp. 110–141. ISSN: 0730-0301. DOI: 10.1145/22949.22950. URL: https://doi.org/10.1145/22949.22950.

[19] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. "Show Me: Automatic Presentation for Visual Analysis". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1137–1144. DOI: 10.1109/TVCG.2007.70594.

[20] *New York City Airbnb Open Data*. https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data.

[21] Devin Petersohn et al. "Towards Scalable Dataframe Systems". In: *Proceedings of the VLDB Endowment* 13.11 (2033-2046), pp. 727–738. DOI: https://doi.org/10.14778/3407790.3407807.

[22] M. Sedlmair et al. "A Taxonomy of Visual Cluster Separation Factors". In: *Computer Graphics Forum* 31.3pt4 (2012), pp. 1335–1344. DOI: https://doi.org/10.1111/j.1467-8659.2012.03125.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03125.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03125.x.

[23] Jinwook Seo and B. Shneiderman. "A Rank-by-Feature Framework for Unsupervised Multidimensional Data Exploration Using Low Dimensional Projections". In: *IEEE Symposium on Information Visualization*. 2004, pp. 65–72. DOI: 10.1109/INFVIS.2004.3.

[24] Jonathan Shi and Connor McMahon. "Leveraging User Think Time to Enable Query Optimizations". In: (2021). URL: https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F21/projects/reports/project4_report.pdf.

[25] Tarique Siddiqui et al. "Effortless Data Exploration with Zenvisage: An Expressive and Interactive Visual Analytics System". In: *Proc. VLDB Endow.* 10.4 (Nov. 2016), pp. 457–468. ISSN: 2150-8097. DOI: 10.14778/3025111.3025126. URL: https://doi.org/10.14778/3025111.3025126.

[26] Phanwadee Sinthong et al. "DQDF: Data-Quality-Aware Dataframes". In: *Proc. VLDB Endow.* 15.4 (Dec. 2021), pp. 949–957. ISSN: 2150-8097. DOI: 10.14778/3503585.3503602. URL: https://doi.org/10.14778/3503585.3503602.

[27] C. Stolte, D. Tang, and P. Hanrahan. "Polaris: a system for query, analysis, and visualization of multidimensional relational databases". In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 52–65. DOI: 10.1109/2945.981851.

[28] Chris Stolte, Diane Tang, and Pat Hanrahan. "Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases". In: *IEEE Trans. Vis. Comput. Graph.* 8 (2002), pp. 52–65.

[29] Manasi Vartak et al. "SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics". In: *Proc. VLDB Endow.* 8.13 (Sept. 2015), pp. 2182–2193. ISSN: 2150-8097. DOI: 10.14778/2831360.2831371. URL: https://doi.org/10.14778/2831360.2831371.

[30] Manasi Vartak et al. "Towards visualization recommendation systems". In: *Acm Sigmod Record* 45.4 (2017), pp. 34–39.

[31] Kanit Wongsuphasawat et al. "Towards a General-Purpose Query Language for Visualization Recommendation". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. HILDA '16. San Francisco, California: Association for Computing Machinery, 2016. ISBN: 9781450342070. DOI: 10.1145/2939502.2939506. URL: https://doi.org/10.1145/2939502.2939506.

[32] Kanit Wongsuphasawat et al. "Voyager 2: Augmenting Visual Analysis with Partial View Specifications". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 2648–2659. ISBN: 9781450346559. DOI: 10.1145/3025453.3025768. URL: https://doi.org/10.1145/3025453.3025768.

[33] Kanit Wongsuphasawat et al. "Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations". In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (2016), pp. 649–658. DOI: 10.1109/TVCG.2015.2467191.

[34] Doris Xin et al. "Enhancing the Interactivity of Dataframe Queries by Leveraging Think Time". In: *IEEE Data Eng. Bull.* 44.1 (2021), pp. 66–78. URL: http://sites.computer.org/debull/A21mar/p66.pdf.

[35] Matei Zaharia et al. "Spark: Cluster computing with working sets." In: *HotCloud* 10.10-10 (2010), p. 95.