THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

View Recommendation for Visual Data Exploration

Humaira Ehsan

Masters of Computer Science

A thesis submitted for the degree of Doctor of Philosophy at The University of Queensland in 2019

School of Information Technology & Electrical Engineering

Abstract

The widespread use of digital systems has resulted in an exponential increase in the volume of available data, with data being collected across all domains; from banking and finance to health and energy. However, this data is only beneficial if it is unlocked to generate insights. Visual data exploration plays an important role in this process of discovering insights. Typically, it involves an analyst going through the following steps: 1) selecting a subset of data for analysis, 2) generating different visualizations of that analyzed data, and 3) sifting through those visualizations for the ones which reveal interesting insights. Based on the outcome of the last step, the analyst might have to refine their initial selection of data so that the newly analyzed subset would show more interesting insights. This is clearly an iterative process, where each selection of data (i.e., *input query*) is a springboard to the next one. For this time-consuming process to be effective, a challenging combination of system and domain expertise is required.

Motivated by the need for an efficient and effective visual data exploration process, several solutions have been proposed towards automatically finding and recommending interesting data visualizations (i.e., steps 2 and 3 above). The main idea underlying those solutions is to automatically generate all possible *aggregate views* of data, and recommend the top-k interesting views, where an interestingness of a view is quantified according to some utility function. Recent work provides strong evidence that a *deviation-based* formulation of utility is able to provide analysts with interesting visualizations which highlight some of the particular trends of the analyzed datasets. In particular, the deviation-based metric measures the distance between the probability distribution of a specific dataset under analysis, called *target view*, and that of a reference dataset, called *comparison view*. The underlying premise is that visualizations with higher deviations are expected to reveal insights which are very particular to the analyzed dataset. While the deviation-based notion of utility has been shown to be effective in recommending views with categorical dimensional attributes, in this work we argue that it falls short in capturing the requirements of numerical dimensions. Furthermore, such visualizations entail high data processing costs because a large number of views are generated to evaluate their usefulness.

In this thesis, we propose novel view recommendation schemes, which incorporate a hybrid multi-objective utility function that captures the impact of numerical dimension attributes. These schemes help to address the challenges associated with high data processing cost and the presence of numerical dimensional attributes. The first scheme, *Multi-Objective View Recommendation for Data Exploration (MuVE)*, adopts an incremental evaluation of our multi-objective utility function, which allows pruning of a large number of low-utility views and avoids unnecessary objective evaluations. The second scheme, upper MuVE (uMuVE), further improves the pruning power by setting the upper bounds on the utility of views and allowing interleaved processing of views, at the expense of increased memory usage. Finally, the third scheme, Memory-aware uMuVE (MuMuVE), provides pruning power close to that of uMuVE, while keeping memory usage within a specified limit.

Moreover, existing solutions have been shown to be effective in recommending interesting views under the assumption that the analyst is precise in their selection of analyzed data (i.e., step 1 above).

That is, the analyst is able to formulate a well-defined input query that selects a subset of data, which contains interesting insights that can be revealed by the recommended visualizations. Such an assumption is clearly impractical and severely limits the applicability of those solutions. In reality, it is typically a challenging task for an analyst to select a subset of data that has the potential of revealing interesting insights. Hence, it is a continuous process of trial and error, in which the analyst keeps refining their selection of data manually and iteratively until some interesting insights are revealed. In this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. Motivated by the need for a query refinement solution that is able to automatically modify the analyst's initial input query into a new query, we propose efficient *Query Refinement for View Recommendation (QuRVe)* schemes, which automatically refine an input query to search for subsets of data having interesting views and recommend the top-k views. However, uncontrolled refinement of queries can lead to problems such as dissimilar refined queries from input query and statistically insignificant results. Therefore, a multi-objective function is proposed to measure similarity, interestingness and significance of the refined queries and their corresponding views. The principle idea underlying proposed QuRVe scheme is to incrementally access the refined queries in order of their similarity with the original query, which allows an early termination of search and results in pruning of a large number of views. Additionally, uQuRVe scheme further reduces the cost by tightening the upper bounds on the utility of the views and short circuiting unnecessary views. Extensive experimental evaluations also support the significant gains provided by our proposed schemes.

Furthermore, the recommended aggregate views are based on the target views defined on subsets of data selected by the refined queries and comparison views defined on the reference dataset The interesting views revealed in such a manner can be considered as *global interesting views*. However, in this work we argue that the assumption in the existing systems that the comparison views are always defined on a specific reference dataset, which is typically the complete database, limits discovery of interesting insights. Such systems fail to discover *local interesting views*, in which the comparison views belong to subsets of data instead of the specified reference dataset. This discovery of local interesting views is a non-trivial task as it increases the search space of views exponentially and more importantly, may combinations of comparison and target views for aggregate views lack contextual relationship with each other. We formulate the problem of *refinement of reference dataset* for view recommendation and propose a baseline scheme to recommend locally interesting views. Moreover, in this thesis we also present the design and implementation of a holistic prototype system *View-360* for view recommendation. Additionally, we also showcase the effectiveness of our schemes on real world datasets along with in depth analysis for insights.

Declaration by author

Publications during candidature

- Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis: *Muve: Efficient multi-objective view recommendation for visual data exploration.* In the Proceedings of 32nd IEEE International Conference on Data Engineering (ICDE), May 2016.

- Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis: *Efficient recommendation of aggregate data visualizations.* IEEE Transactions on Knowledge and Data Engineering, 30, 2, 2018.

- Humaira Ehsan and Mohamed A. Sharaf: *Materialized View Selection for Aggregate View Recommendation.* In the Proceedings of Australasian Database Conference (ADC), January 2019.

Submitted manuscripts included in this thesis

- Humaira Ehsan, Mohamed A. Sharaf and Gianluca Demartini: *QuRVe: Query Refinement for View Recommendation in Visual Data Exploration.* Submitted to Submitted to 35th ACM Symposium On Applied Computing (ACM(SAC)), Mar 2020.

Publications included in this thesis

The following publication has been incorporated as Chapter 3.

1. [1] **Humaira Ehsan**, Mohamed A. Sharaf, and Panos K. Chrysanthis, Muve: Efficient multi-objective view recommendation for visual data exploration, In the Proceedings of 32nd IEEE International Conference on Data Engineering (ICDE), May 2016.

Contributor	Statement of contribution	%
Humaira Ehsan (candidate)	Problem Formulation	80
	Design of Algorithms	80
	Experiments	100
	Paper Writing	60
	Proof Reading and Feedback	50
Mohamed A. Sharaf	Problem Formulation	20
	Design of Algorithms	20
	Paper Writing	30
	Proof Reading and Feedback	30
Panos K. Chrysanthis	Paper Writing	10
	Proof Reading and Feedback	20

2. [2] **Humaira Ehsan**, Mohamed A. Sharaf, and Panos K. Chrysanthis, Efficient recommendation of aggregate data visualizations,IEEE Transactions on Knowledge and Data Engineering, 30, 2, 2018.

Contributor	Statement of contribution	%
Humaira Ehsan (candidate)	Problem Formulation	80
	Design of Algorithms	100
	Experiments	100
	Preparation of Figures	100
	Paper Writing	70
	Proof Reading and Feedback	50
Mohamed A. Sharaf	Problem Formulation	20
	Paper Writing	20
	Proof Reading and Feedback	30
Panos K. Chrysanthis	Paper Writing	10
	Proof Reading and Feedback	20

3. [3] **Humaira Ehsan** and Mohamed A. Sharaf: Materialized View Selection for Aggregate View Recommendation, In the Proceedings of Australasian Database Conference (ADC), January 2019.

Contributor	Statement of contribution	%
Humaira Ehsan (candidate)	Problem Formulation	80
	Design of Algorithms	100
	Experiments	100
	Preparation of Figures	100
	Paper Writing	80
	Proof Reading and Feedback	70
Mohamed A. Sharaf	Problem Formulation	20
	Paper Writing	20
	Proof Reading and Feedback	30

The following publication has been incorporated as Chapter 4.

1.**Humaira Ehsan**, Mohamed A. Sharaf and Gianluca Demartini: QuRVe: Query Refinement for View Recommendation in Visual Data Exploration. Submitted to 35th ACM Symposium On Applied Computing (ACM(SAC)), Mar 2020.

Contributor	Statement of contribution	%
Humaira Ehsan (candidate)	Problem Formulation	80
	Design of Algorithms	100
	Experiments	100
	Preparation of Figures	100
	Paper Writing	70
	Proof Reading and Feedback	70
Mohamed A. Sharaf	Problem Formulation	20
	Paper Writing	20
	Proof Reading and Feedback	20
Gianluca Demartini	Paper Writing	10
	Proof Reading and Feedback	10

Contributions by others to the thesis

My principle advisor, Dr. Mohamed Sharaf, has largely contributed towards the research problems presented in this thesis. Dr. Sharaf assisted me by providing guidance and feedback on formulating the problems and solutions in this thesis. He also reviewed, polished and assisted with the published papers included as part of this thesis.

Statement of parts of the thesis submitted to qualify for the award of another degree

No works submitted towards another degree have been included in this thesis.

Research Involving Human or Animal Subjects

No animal or human subjects were involved in this research.

Acknowledgments

My PhD work would not have been possible without the support of an incredible number of people who have helped me navigate this path, and believed in me.

First and foremost, this PhD was impossible without the unwavering support of my PhD advisor and mentor, Dr. Mohamed Sharaf. I would like to express my sincere gratitude to Sharaf for sharing his immense knowledge about research and for pushing me to my limits to improve. Mohamed's patience, support and trust during this time has been instrumental in the completion of this thesis.

I also want to thank Professor Shazia Sadiq, and Dr. Gianluca Demartini for taking the time out of their busy schedules to serve on my advisory committee. Your encouragement, and insightful comments are greatly appreciated. I want to thank the co-author of my papers: Panos K. Chrysanthis for his support, knowledge sharing and valuable feedback.

This PhD and everything else that I have achieved would not have been possible without the prayers, love, and motivation of my parents. My husband, Ehsan, has been with me every day of this journey and has many stories to tell! His encouragement, support, and patience kept me going. To my beloved daughter Sualyha, I would like to express my thanks for being such a good girl always cheering me up.

My PhD would not have been possible without the PhD wisdom of my dear friends, Amina and Hina. My PhD experience would have been so much less fun and enriching without them. My brother, Hammad, has been yet another rock of support for me and a place I have turned to for advice and encouragement so many times. I also want to thank all my colleagues and friends at Data and Knowledge Engineering (DKE) group for their support.

Financial support

This research was supported by IPRS Commonwealth Scholarship and Australian Government Research Training Program Scholarship.

Keywords

Visual Data Exploration, View Recommendation, Query Refinement

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080604, Database Management, 100%

Fields of Research (FoR) Classification

FoR code: 0806, Information Systems, 100%

Dedicated to ...

All the women out there working hard to get ahead ...

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Overview

The amount of data being generated and collected daily is in the order of terabytes. However, this data is only beneficial if it is unlocked to generate insights. Visual data exploration plays an important role in discovering *interesting insights*. A typical data exploration process involves the analyst systematically engaging in activities as shown in Figure 1.1. First, the analyst chooses a *subset* of data that might be of interest by writing an *input query* or by using some graphical interface. Second, she analyses that subset for discovering interesting insights, using various techniques such as tabular data, graphical representations, statistical summaries etc. Finally, she refines the subset selection by modifying the input query of first step and then repeating the process till some insights are revealed. This clearly is a hit and miss process through which discoveries about the data are made. The effectiveness of this process depends on the ability of the analyst, performance of the exploration techniques and the complexity of the dataset. For this time-consuming process to be effective, a challenging combination of system and domain expertise is required. Particularly, selecting a subset of data that would reveal interesting insights requires domain knowledge and a comprehensive knowledge about the dataset. Additionally, for complex data, it is infeasible for an analyst to manually generate and browse all possible visualizations for insights. Therefore, there is a need for *automated solutions* that can effectively *recommend* such visualizations.

Motivated by the need for an efficient and effective visual data exploration process, several solutions have been proposed towards automatically finding and recommending interesting data visualizations [6–12]. The main idea underlying those solutions is to automatically generate all possible visualizations, and recommend the *top-k* interesting visualizations, where an *interestingness* of a view is quantified according to some *utility function*. However, such solutions come at the expense of high data processing costs, where a large number of visualizations are generated and their interestingness is evaluated. Moreover, the hardest part of visualization recommendation is to quantify what would be interesting for the analyst. Consequently, the visualization recommendation problem has been approached from different angles, such as deviation-based/similarity-based methods that

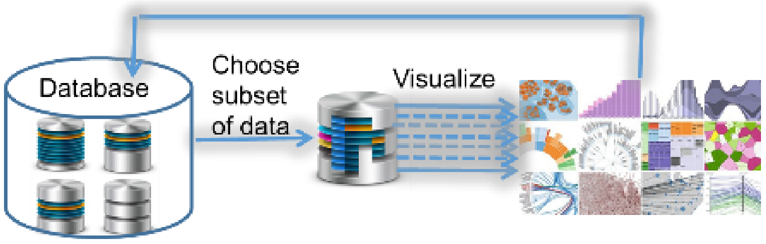


Figure 1.1: Data exploration process- Key steps

quantify interestingness as a similarity/distance metric [6, 13–15], user-actions-based methods that quantify interestingness in term of user's intent, which is inferred by her present actions or by historic data [7, 16], perception-based methods, that learn human perception and use it to mine and recommend interesting visualizations [10–12].

In particular, recent work provides strong evidence that a *deviation-based* formulation of utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets. Specifically, the deviation-based metric measures the distance between the *probability distribution* of the subset of data to be analyzed called the *target dataset* and that of a *reference dataset*, which is typically the entire database from which that target dataset is extracted. Moreover, the visual representation of these probability distributions is called an *aggregate view*, which is then plotted using some of the popular visualization methods (e.g., bar charts, scatter plots, etc.). The underlying premise is that a visualization that results in a higher deviation is expected to reveal some interesting insights that are very particular to the analyzed dataset [6, 17]. The problem of recommending top-k aggregate views from deviation-based utility metric is formally stated as:

Definition 1. Top-k View Recommendation: *Given a database D_B, a user-specified query Q which selects a subset D_Q from database D_B, a deviation based utility function U, and a positive integer k,* **find the** *k* **aggregate views over** *D_Q, which have the highest deviation values.*

The baseline solution to the problem defined in Def. 1 is that all possible aggregate views are generated, deviation of each view is calculated and the the top-k views with highest deviation values are recommended. However, while the deviation-based notion of utility has been shown to be effective in recommending views with categorical dimensional attributes, in this thesis we argue that it falls short in capturing the requirements of numerical dimensions [18, 19]. Particularly, in the presence of such numerical dimensions, *binned aggregation* is required to group the numerical values along a dimension into adjacent intervals. Given the large number of options for binning a numerical dimension, it is expected that different binning configurations will result in different deviations, and in turn, different levels of interestingness from the analyst point of view. For instance, in a view with small number of bins, interesting insights are expected to remain hidden under a smooth and coarse visual representation. Meanwhile, in a view that contains a large number of bins, insights might go unnoticed in a cluttered or sparse visualization. Hence, the problem defined in Def. 1 is extended for view recommendation in the presence of numerical dimensions as follows:

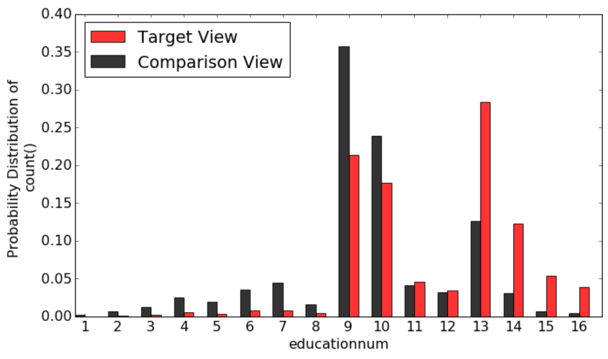


Figure 1.2: Non-Binned Aggregate View
Target view $V_i(D_Q)$ (income>50K) and **Comparison view** $V_i(D'_Q)$ (income≤50K)
Deviation= 0.17866330071272257

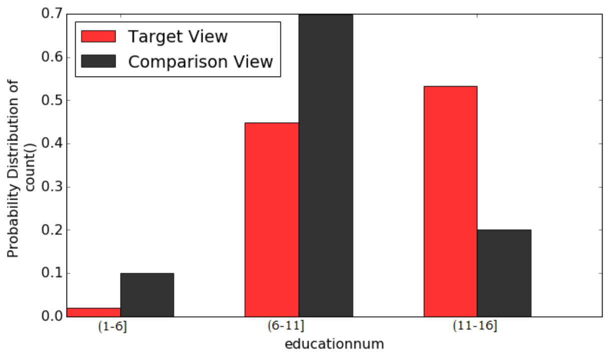


Figure 1.3: Binned Aggregate View
Target view $V_i(D'_Q)$ (income>50K) and **Comparison view** $V_i(D'_Q)$ (income≤50K)
Deviation= 0.2993347659872509

Definition 2. View Recommendation for Numerical Dimensions: *Given a database D_B, a user-specified query Q which selects a subset D_Q from database D_B, a deviation based utility function U, and a positive integer k, find the k* **binned** *aggregate views over D_Q, which have the highest deviation values.*

To illustrate the impact of binning on numerical dimensions, consider the following example:

Example 1. *Consider an analyst who wants to explore and find interesting insights in the U.S. Census income dataset [20], which is stored in table C. She notices that the categorical attribute* `income` *has two categories, that is* `income` $>$ `50K` *and* `income` $\leq$ `50K`. *Her intuition is that analyzing the two categories of income might reveal some interesting insights. Therefore, for her analysis, she chooses the target dataset (D_Q) consisting of those individuals who have* `income` $>$ `50K`, *by posing the following input query Q:*

`Q: SELECT * FROM C WHERE income` $>$ `50K`,

Similarly, for reference dataset, she chooses the individuals with `income` $\leq$ `50K`, *which is complement of D_Q. Both target and reference datasets include dimension (e.g., gender, education, occupation, etc.), and measure (e.g., final wages, capital gain, etc.) attributes.*

To recommend interesting bar chart visualizations, different SQL aggregate functions are applied on the views resulting from all the possible pairwise combinations of dimensions and measures, then *the most interesting views are presented to the analyst. Fig. 1.2 shows one of the top-k visualizations defined on the dimension* `years of education (educationnum)` *with aggregate function* `COUNT`. *Particularly, such visualization is equivalent to plotting the probability distributions of a view $V_i(D_Q)$ defined on target dataset (*target view*) and a View $V_i(D'_Q)$ defined on reference dataset (*comparison view*):*

$V_i(D_Q)$:`SELECT educationnum, COUNT(*) FROM C`
`WHERE income > 50K GROUP BY educationnum`

$V_i(D'_Q)$: `SELECT educationnum, COUNT(*) FROM C`
`WHERE !(income > 50K ) GROUP BY educationnum`

Hence, the deviation value shown in Fig. 1.2 is the distance between the probability distribution of $V_i(D_Q)$ and $V_i(D'_Q)$. At first glance, comparing the two views fails to reveal clear insights about compared income groups. However, binning the two views, as shown in Figure 1.3, reveals some very interesting observation. Particularly, Figure 1.3 shows that more than 50% of individuals having `income > 50K` *have college education (i.e., educationnum$\geq$12). However, 70% of the individuals having* `income`$\leq$`50K` *have education between 6-11 years. Clearly, this observation reflects that generally people having* `income`$\leq$`50K` *did not get college education.*

As the above example shows, choosing the right binning is essential in the process of extracting insights from the data, whether that process is performed manually or analytically. In Chapter 3, we address the challenges introduced by binning. We propose a multi-objective function and novel suite of search schemes for efficient recommendation of top-k aggregate data views for numerical dimensions.

Moreover, the proposed solutions [1–3, 6] are shown to be effective in recommending interesting views under the assumption that the analyst is precise in their selection of analyzed data. That is, the analyst is able to formulate a well-defined input query that selects a subset of data, which contains interesting insights that can be revealed by the recommended visualizations. In reality, it is typically a challenging task for an analyst to select a subset of data that has the potential of revealing interesting insights. Hence, it is a continuous process of trial and error, in which the analyst keeps refining their selection of data manually and iteratively until some interesting insights are revealed. Therefore, in this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. That is, there is a need for solutions in which the two tasks of data selection as well as view recommendation are both automated and work together in synergy. Hence, the problem formulated in Def. 2 can be expanded to include the above mentioned tasks of data selection as:

Definition 3. Input Query Refinement for View Recommendation: *Given a database D_B, a user-specified query Q which selects a subset D_Q from database D_B, a deviation based utility function U, and a positive integer k,* **automatically refine** *Q, to generate a set $\mathbb{Q}$ of all possible refined queries Q_j such that $Q_j \in \mathbb{Q}$ and find the k aggregate views over all D_{Q_j}, which have the highest deviation values.*

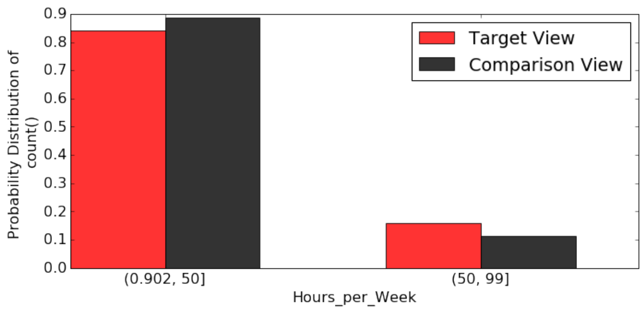


Figure 1.4: View from Input Query Q (Lacks Deviation)
Target View $V_i(D_Q)$**:** `SELECT Hours_per_Week COUNT(*) FROM C WHERE education` $\geq$ `12 GROUP BY Hours_per_Week`
Comparison View $V_i(D_B)$**:** `SELECT Hours_per_Week COUNT(*) FROM C GROUP BY Hours_per_Week`
Deviation = `0.04598937500117262`

To further illustrate the need for such solution, consider the following example.

Example 2. *Consider Ex. 1 again, now the analyst knows that there is something interesting about having higher education. Therefore, she decides to perform further analysis on subset of data of those who have achieved a high level of education. Hence, she selects from the overall Census data that particular subset in which everyone has completed their 12th year of education (i.e., graduated high school) via the the following input query:*
`Q: SELECT * FROM C WHERE education` $\geq$ `12`,

To find top-k visualizations, she might use one of the existing approaches (e.g., [1, 6]), in which the target and comparison views are generated and their deviation is computed by using a distance function. Fig. 1.4 shows one of the top-k visualizations recommended by such approaches. Particularly, the figure shows a bar chart in which the x-axis is defined on dimension `Hours per week`*, and the y-axis is the probability distribution of the aggregate function* `COUNT`.

Moreover, it can be clearly seen from Fig. 1.4 that the target and comparison views are almost the same, which is also reflected by the low-deviation value (i.e., deviation = 0.045989..). However, such visualization would still be recommended by existing approaches because it achieves the maximum deviation among all the views generated over the data subset selected by query Q, despite of that maximum value being inherently low. That is, the subset of data selected by the analyst pertaining to those who completed their 12th year of education falls short in showing any interesting insights.

The previous example illustrates a clear need for a query refinement solution that is able to automatically modify the analyst's initial input query into a new query, which selects a subset of data that includes interesting insights. Those hidden insights are then easily revealed using existing solutions which are able to recommend interesting visualizations. To that end, one straightforward and simple approach would involve generating all the possible subsets of data by automatically refining all the predicates of the input query. Consequently, for each subset of data selected by each query refinement, generate all possible aggregate views. In addition to the obvious challenge of a prohibitively large search space of query refinements, this naive approach would also lead to views that might appear

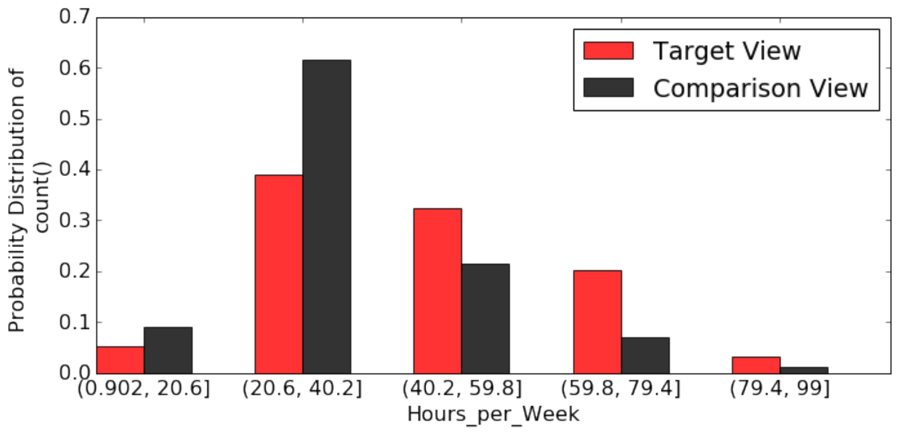


Figure 1.5: View from Refined Input Query and Refined Reference Dataset
Target View $V_i(D_{Q_1})$**:** `SELECT Hours_per_Week COUNT(*) FROM C WHERE education` $\geq$ `14 GROUP BY Hours_per_Week`
Comparison View $V_i(D_{Q_2})$**:** `SELECT Hours_per_Week COUNT(*) FROM C WHERE education` $\leq$ `5 GROUP BY Hours_per_Week`
Deviation = `0.23288980694954403`

to be visually interesting but they are useless from the analyst's perspective. Particularly, the naive approach might lead to refined queries which are significantly dissimilar from the input query and recommend views that are statistically insignificant. In Chapter 4, we formulate our problem of query refinement for view recommendation, propose a multi-objective function and constraints to measure similarity, interestingness and significance of the refined queries and their corresponding views. Moreover, novel algorithms are proposed for the efficient navigation of the refined queries search space for recommendation of data visualizations.

As mentioned earlier, in the recommended aggregate views, the target views are defined on subsets(target datasets) selected by refined queries, while comparison views are defined on a reference dataset. Particularly, the reference dataset can be the complete database as in Ex. 2 or complement of the target dataset as in Ex. 1. Hence, the interesting views revealed in such a manner can be considered as *global interesting views*, as the aggregate view generation involves the complete database (global) one way or another. However, contrary to global interesting views, there can exist *local interesting views* where the comparison views come from another subset of data instead of the given reference dataset. In particularly, that means refining the reference dataset as well. Consequently, all possible aggregate views, include all combinations of target views corresponding to all target datasets (i.e., refined queries) and comparison views from all refined reference datasets. Therefore, the problem defined in Def. 3 can be extended and redefined as:

Definition 4. Reference Dataset Refinement for View Recommendation: *Given a database D_B, a user-specified query Q which selects a subset D_Q from database D_B, a deviation based utility function U, a positive integer k, and a set of automatically refined queries $\mathbb{Q}$, automatically* **refine the reference dataset** *and find the k global and local aggregate views, which have the highest deviation values.*

To illustrate the impact of refinement of reference dataset, consider the following example:

Example 3. *Consider the U.S. Census income dataset [20] and the input query of Ex. 2.*

The view shown in Figure 1.5 is the top-1 view after the refinement of the input query and the refinement of the reference dataset. Particularly, this aggregate view is generated by computing the deviation between the probability distribution of a target view define on the refine query Q_1 and a comparison view defined on a refined reference dataset which is selected by query Q_2.

Q_1:`SELECT * FROM C WHERE WHERE education` $\geq$ `14`

Q_2: `SELECT * FROM C WHERE WHERE WHERE education` $\leq$ `5`

Note that the view shown in in Figure 1.5 is more interesting than the view of Figure 1.4 and it is also reflected by the deviation value. However, it is a locally interesting view where two subsets of data are compared and insight is revealed about those two subsets.

The previous example illustrates clearly that refining the reference dataset can lead to more interesting views. A naive approach would be to generate all possible combinations of target and comparison views from refined queries and refined reference dataset and then recommend the top-k views. In addition to a prohibitively large search space, the naive approach would also lead to views that lack semantic value. Particularly, every possible combination of target and comparison view might lead to aggregate views in which the compared comparison and target view has no contextual connection to each other. In Chapter 5 we formulate the problem of reference dataset refinement for view recommendation and include a baseline scheme. We also present a prototype system View-360 that recommends top-k aggregate view after considering all aspects of view recommendation.

Next, we present the contributions of this thesis.

1.2 Thesis Contribution

Motivated by the need to efficiently and effectively recommend aggregate views for visual data exploration, we have addressed multiple aspects of view recommendation problem: 1) Recommendation of top-k aggregate data views in the presence of numerical dimensions. 2) Automatically refining the user's initial input query and the reference dataset query to get to the aggregate visualizations that are interesting. 3) View-360 a prototype system for aggregate view recommendation for visual data exploration. 4) Effectiveness based analysis of two real world datasets.

1.2.1 Efficient View Recommendation for Numerical Dimensions

To address these challenges of view recommendation in the presence of numerical dimensions, we introduce a novel hybrid *multi-objective utility* function, which captures the impact of numerical dimension attributes in terms of generating visualizations that are: 1) interesting, 2) usable, and 3) accurate. Combining these often conflicting objectives dramatically expands the search space of possible visualizations (i.e., aggregate views). Moreover, it significantly increases the processing time incurred to asses the overall utility of each view, which is assembled from the utility values of each of the three objectives listed above.

Therefore, in this work, we present a novel suite of search schemes for efficient recommendation of top-k aggregate data views.

Multi-Objective View Recommendation for Data Exploration (MuVE): We propose the Multi-Objective View Recommendation for Data Exploration schemes [1, 2]. The main idea underlying our first scheme *Multi-Objective View Recommendation for Data Exploration (MuVE))* is to use an incremental evaluation of the multi-objective utility function, where different objectives are computed progressively. Our results in [1] show that MuVE is able to prune a large number of unnecessary views, and in turn reduces the overall processing time for recommending the top-k views. However, the pruning power is highly dependent on the order in which the views are presented to MuVE and might often limit its performance gains.

upper MuVE (uMuVE): To address the limitation of MuVE, we propose our second scheme *upper MuVE (uMuVE)*, in which the goal is to provide a flexible navigation of the search space so that high-utility views are discovered earlier. Particularly, uMuVE is based on setting upper bounds on the utility of each possible view, which is then exploited to effectively guide the search process by means of interleaving the evluation of the different objectives offered by the different views. Due to that interleaved processing, at any point of time, uMuVE would typically have multiple views under consideration, which requires significant amount of memory for storing their data.

Memory-aware uMuVE (MuMuVE): The improvement in uMuVE comes at the expense of high memory usage. We propose another scheme MuMuVE that aims to provide a pruning power close to that of uMuVE, while keeping memory usage within predefined constraint.

Materialized View Selection for Aggregate View Recommendation (mView): The most expensive operation while computing the utility of views is the time spent in executing the queries related to the views. To reduce the cost of this particular operation, we propose a novel technique mView , which instead of answering each query related to a view from scratch, reuses results from the already executed queries. In particular, this is done by materializing views and answering queries from the materialized views instead of the base table. Due to prohibitively large number of views, the blind application of materialization may result in even further degradation of the cost. In this work we first defines a cost benefit model to decide which views are the best to reuse. Later, we propose scheme mView which materializes the best set of views in an optimal order and consequently reduces the overall cost of the solution [3]. After analyzing the trade off between cost savings of mView and MuVE scheme, we also propose extension of mView for MuVE Scheme.

1.2.2 Query Refinement for View Recommendation

In this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. Particularly, we highlight the need for automatic refinement solutions that are guided by the user's preference. Consequently, we propose a novel suite of schemes for automated query refinement for view recommendation in visual data exploration.

Query Refinement for View Recommendation (QuRVe): The main idea underlying our QuRVe schemes is to incrementally access the refined queries in order of their similarity with the original query, which allows an early termination of search and results in pruning of a large number of views.

upper Query Refinement for View Recommendation (uQuRVe): The upper bound on deviation used by QuRVe scheme to prune low utility views is a theoretical extreme value and it is oblivious to the under consideration target and comparison view.The main idea of uQuRVe scheme is to provide tighter upper bound on deviation of views by using the properties of deviation function and the already executed comparison views.

upper Query Refinement for View Recommendation - range(uQuRVe-range): The main idea behind uQuRVe-range is to reduce search time even further by having more control on the order in which views are generated. Particularly, it prioritize to generate the views with the high utility first.

Query Refinement for View Recommendation (QuRVe)-Approximation: Approximation based extension for uQuRVe and uQuRVe-range schemes are proposed to further improve performance, while incurring negligible loss in the quality of recommendation.

1.2.3 View-360: A Prototype System for View Recommendation

In this work, we argue that in existing recommendation systems, the assumption that a comparison view is generated from a default or predefined reference dataset, limits the discovery of interesting insights. Therefore, the reference dataset should also be refined by refining the query that selects the reference dataset, we name it the reference dataset refinement. The aggregate view recommendation with the refinement of reference dataset is non-trivial as it increases the search space of views exponentially, and more importantly all combinations of comparison views and target views in an aggregate view are not comparable due to the lack of contextual connection between them. In this thesis, we define the problem of reference dataset refinement for view recommendation and present a baseline scheme.

View-360: The design and implementation of a holistic prototype system **View-360** for view recommendation is presented. Particularly, it includes all of the search schemes presented in this thesis. Moreover, it also includes all aspects of aggregate view recommendation i.e, recommendation based on categorical and numerical attributes, and recommendation based on refinement on target and comparison queries.

Dataset analysis: We showcase the effectiveness of all of our proposed schemes in this thesis by performing detailed analysis on real datasets from various domains. Particularly, we show all steps of data analysis and insights discovery on two datasets: one from general domain and one from the health domain.

1.3 Thesis Layout

The rest of the thesis is organized as follows: In Chapter 2, we present preliminaries and related work. In Chapter 3, we present a suite of search schemes for efficient recommendation of top-k aggregate data views. In Chapter 3.5, we present the schemes to solve the problem of materialized view selection for binned aggregate views. In Chapter 4, we present a suite of efficient schemes that automatically refine input query. In Chapter 5, we present the problem of reference dataset refinement and we show application of our schemes on real world dataset from different domains. Finally, Chapter 6 concludes this thesis and overviews future work.

Chapter 2

Related Work

2.1 Data Exploration

Traditionally databases have structures, precise data, clear input queries and specific users, makes data analysis easy and efficient. However, with the revolution of Big Data we are forced to rethink our theories and implementation of databases [21]. Now almost every domain has a huge volume, velocity, variety and veracity of data. Addtionally, users with varying level of technical skills and domain knowledge are performing analysis tasks. Therefore, there have emerged many new facets of storage, retrieval, presentation and exploration of big data.

Particularly, data exploration can be performed in two ways:

- *Open ended exploration*: Data characteristics are hidden from the user and exploration can mean finding interesting insights in data, investigating and seeking inspiration, and suggesting new hypothesis.
- *Targeted exploration*: User is somewhat familiar with data characteristics and exploration is used as a tool to evaluate the quality of the data, compare specific data, use data to make decisions, and verify existing hypothesis.

Both open ended and targeted data exploration pose challenges that cannot be addressed through traditional search and query mechanisms [22, 23]. Traditional data search relies heavily on the

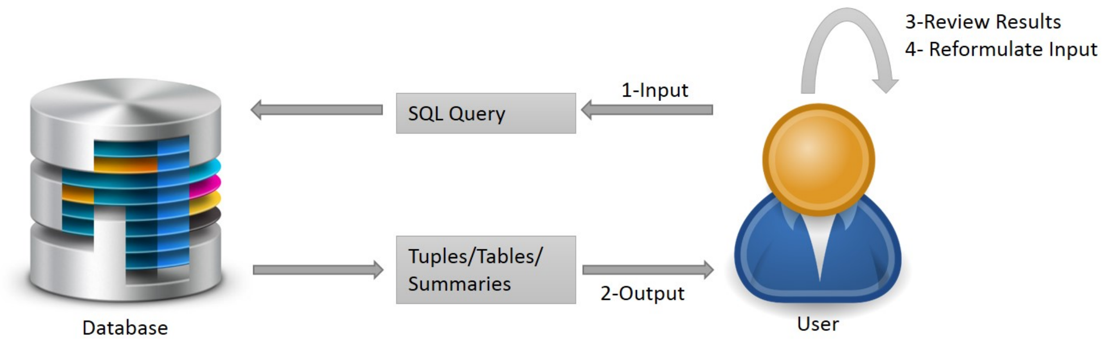


Figure 2.1: Data exploration

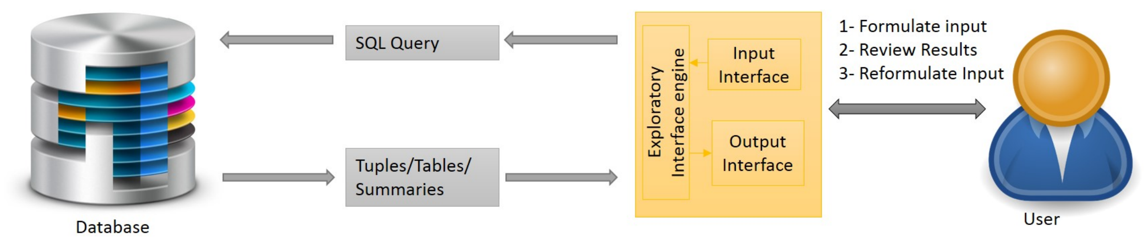


Figure 2.2: Data exploration Interface

database structure and built in queries. In contrast to traditional data search, any data exploration process includes several iterations of following steps, as show in Figure 2.1:

1. issuing a query to the database
2. executing query and generating results
3. reviewing results
4. reformulating the next query

Although database is the key component of this exploration process, only step 2 of executing query and generating results is database-centric. Whereas steps 1, 2 and 3 are user-centric. Hence, the efficiency and effectiveness of the exploration process primarily depends on the user. The assumption of traditional database systems, that the users can formulate structured queries to the underlying database in order to achieve analysis tasks, is often not true for data exploration. Additionally, user-centric processes, are inherently long and laborious which often end in undesired results. Therefore, improving the user interactions with underlying database has been the spotlight of research recently.

There have been many research efforts to develop exploratory interfaces that create a layer between the user and the database as shown in Figure 2.2. It facilitates the user in formulating input, for instance, recent interface allow users to provide input in terms of keywords, uncertain queries, example tuples, graphical parameters and the output can be displayed in the form of interesting visualizations which aid steps of the result review and reformulation of the input for the next iteration. Key requirements of the exploratory interface is that it should be simple enough to avoid complicate declarative languages and, at the same time, it should have flexibility and expressiveness to satisfy complex information needs [24]. Generally, exploratory interfaces can be divided into the following three categories [21]:

1. **Example-driven exploration**: Generally in the data-driven exploration, the user is unable to express their data interests precisely, but she may have an idea of what an interesting result will look like [24–28]. In such situations, the user can be aided with an interface to navigate through subsets of data to find interesting insights. This can be done by showing the user a few samples from the dataset, get feedback and find interesting objects based on presented samples. For instance, AIDE [27, 29] predicts a query that retrieves user's objects of interest. First it prompts the user to label a set of sample objects as relevant or irrelevant, based on this feedback it collects new set of sample objects. AIDE learns the user interests based on his relevance

feedback on strategically collected samples. Ymaldb [30] is another approach which presents the users with additional items that are not part of the results of their original query but may be of interest to them. The computation of such results is based on the frequency of the most interesting (attribute, value) pairs in the user query result and in the database instance. In [28], it is assumed that the user is aware of a few example tuples that should be present in the output of the query and the proposed framework discovers the minimal project join queries based on an example table. These systems at some level require the user to be familiar with the data model they are exploring, which may not be true for the naive user. Therefore, these systems are suitable for a particular set of users and usage scenarios.

2. **Assisted Query Formulation**: Numerous novel query interfaces have been proposed that assist the user in formulating the target queries [31–37]. Particularly, user guided visual tools and query recommendation tools have been developed for specifying relational queries for data exploration. For instance, DataPlay [33] is a sophisticated query specification tool which provides features like graphical specification of query and constraint recommendations for fine tuning the query. Moreover, it also provides a feature to add or remove results and auto correct the query accordingly. Snipsuggest [36] is a tool for non-expert database users, who need to perform complex analysis. As the user starts typing a query, SnipSuggest proposes several completions using relevant snippets collected from a log of past queries. The more a user writes, the more accurate the suggestions get. Another query recommendation system Charles [37] introduces a Segmentation Description Language (SDL). As the user provides a query, Charles breaks its extent into meaningful segments and returns the subsequent SDL descriptions. This provides insight into the set described and offers the user directions for further queries.

3. **Visualization tools**: Visualization has many faces in the context of data exploration. Visualization tools effectively assist the users in formulating input, however, such tools come under the category of assisted query formulation. Visualization also plays a key role in reviewing and interpretation of results for data exploration. Moreover, there are even new types of interactions proposed through visualization such as collaborative annotations and searches [38]. For the purpose of effective and efficient data exploration, automatic recommendation of interesting visualizations has been focus of research lately [6–9, 17], which is focus of this thesis as well. The role of visualization in data exploration and the existing tools and techniques are reviewed in the next section.

2.2 Visual Data Exploration

Data visualization is perhaps the most widely used tool in a data analyst's toolbox [39]. Generally, generating a visualization involves specification of visualization type (e.g., bar chart, scatter plot), filtering (e.g., selecting subset of data to visualize), transformation (e.g., aggregation on data), parameters (e.g., assigning attributes to axes) and visual encodings (e.g., color, size). Range of visualization tools

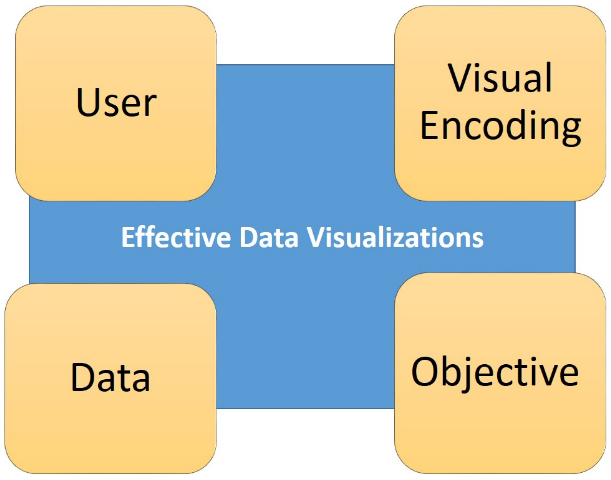


Figure 2.3: Components of Effective Visualization

are available with different features to create these data visualizations. Some of these tools are more expressive, giving expert users more control for instance, ggplot2 , vega-Lite, D3 but require significant programming skills, while others are easier to learn and faster to create visualizations such as Microsoft Excel, Google Spreadsheets but still require mappings between data and visualizations [40].

The following key components are involved in creating an effective visualization (Figure 2.3):

- **Data**: What is the quality of data and what portion of data (tables, tuples, attributes) is visualized.
- **Objective**: What is goal of generating the visualization.
- **Visual Encoding**: What is the type, size, color and other attributes of the visualization, what will be the mapping between attribute of data and axes of visualization
- **User**: What is the level of technical skills and domain knowledge of the user affects, additionally what is the user expectation and preference from the visualizations.

Moreover, generating an effective visualization is highly challenging for big data due to a number of factors such as, quality of data, volume of data, limitation of the tools, time constraints, varying technical expertise and domain knowledge of the user.

The idea of the visual data exploration approach is to iteratively create and refine visualization for the open ended or targeted data exploration goals. Recent years have seen the introduction of many visual analytic tools such as Tableau, Qlik and Spotfire [41–43], particularly for the purpose of visual data exploration. These tools aim to provide aesthetically high-quality visualizations and easy interfaces for the analyst. However, as mentioned before, generating useful visualizations requires complete knowledge about the data, clear objective of visualization, domain knowledge, expertise in statistics and visualization design. *Even super analysts face cognitive barriers in these settings. Time pressures and data overload work against the analyst's ability to rigorously follow effective methods for generating, managing, and evaluating visualizations [44].* Additionally, the exploration process by default has lack of clarity in terms of objective of exploration and for a non-expert user visual data exploration becomes even more challenging and time-consuming. To speed up the data exploration process, the visual exploration tools can be complemented with automated recommendation of interesting visualizations.

2.3 Visualization Recommendation

The goal of visual recommendation is to aid the users in visualization design and in taking exploration decisions by providing quick traversal through the space of visualizations. Particularly visualization recommendation systems automatically generate visualizations, measure the utility of a visualization, and recommend the interesting ones, in terms of the factors mentioned Figure 2.3. In design of visualization recommendation systems, different systems may prioritize different factors depending on the recommendation goals and intended applications. Accordingly the recommendations systems can be mapped to the model displayed in the Figure 2.4. The three corners of the triangle are objective-driven, interaction-driven and data-driven recommendations. The three corners are not mutually exclusive, for instance, a data-driven recommendation need to have some online/offline interaction to initiate the recommendation process and it also has, at the least, some open ended exploration objective. Therefore, the existing systems are around the body of the triangle some are more towards one corner while the others are on the other end. However, we categorize the existing work into one of the three categories based on the contribution of the three categories in the recommendation of visualizations. Below we list down the categories and we briefly mention the systems that belong to each category:

- *Objective-Driven*: Recommendation criteria is based on a particular objective such as missing value, outlier, finding specific pattern, and analysis on particular attributes of data. For instance, Profiler [8] automatically flags problematic data and recommends visualizations based on mutual information metric. Semantic windows [34] is designed to find number of queries that find rectangular regions of the data space the user is interested in. Rank-by-Feature Framework [45] computes statistical summaries and ranking for histograms and scatter-plots.While visualizations can be ranked by various features, the user still selects a ranking criterion according to the task at hand, and then all possible projections are ranked by that criterion.

- *Interaction-Driven*: The recommendation criteria is based on online interaction with the user and the user feedback is used to continuously improve the recommendation. AIDE [29] engages the user in a conversation by getting feedback, while in the background the system builds user model that predicts data matching user interest. INDIANA [22] assists the user in gaining insights through an interactive and incremental process. Statistically grounded algorithms are proposed to support interactive data exploration. ForeCache [46] is a tool for suggesting browsing patterns on multi-dimensional numerical data to users; the recommendations are provided with different levels of granularities by aggregating data, and are strongly focused on the user's interests about previously explored data.

- *Data-Driven*: Recommendation criteria is based on either data characteristics such as data summaries, distributions, correlations, or some historical data, data collected from offline

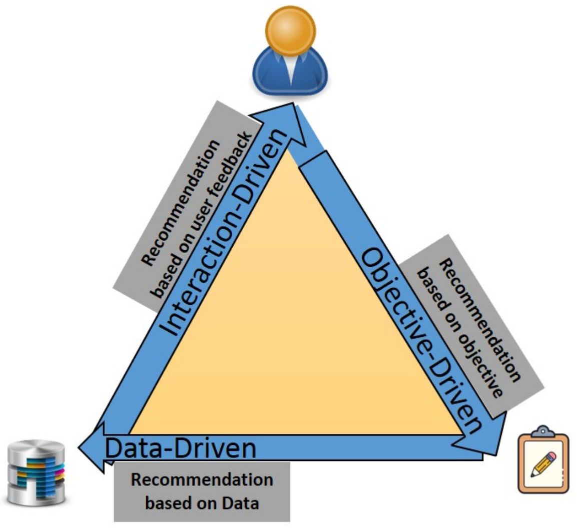

Recommendation based on user feedback
Interaction-Driven
Objective-Driven
Recommendation based on objective
Data-Driven
Recommendation based on Data

Figure 2.4: Visualization Recommendation Systems

interactions. We discuss the features, key issues and systems of this category in detail in the next section.

2.3.1 Data-Driven Recommendation Systems

The main idea of data-driven recommendation systems is to automatically discover and recommend visualizations based on properties of data, with minimal interaction with the user. These systems are most suitable when the user lacks clear objective of exploration and has low level of familiarity with the data. The key issues in designing such systems are:

1. *Quantifying Interestingness*: There is a lack of unified and consistent formulation of interestingness measure. A number of methods have been proposed to quantify intrestingness such as, deviation-based, similarity-based and perception-based [6, 10, 47].

2. *User preferences*: Different users may be interested in different attributes or visualizations. Having user preference included in the recommendation scheme provides an important criteria to evaluate which visualizations are useful for different users [39]. At the same time these systems are targeted to keep the user interaction to the minimum so that users of all skill levels can use them effectively.

3. *Scalability*: Data-driven recommendation system can be computationally expensive as it involves generation and evaluation of huge number of visualization, therefore scalability becomes a challenge. Even for a moderate size dataset the exploration may involve exponential space of visualizations [39].

4. *False Discoveries*: The data-driven approach of recommendation significantly increases the risk of finding false discoveries [48]. Therefore, the insights should be tested for statistical significance.

5. *Coverage*: Covering all parts of dataset with every possible visualization is a non-trivial task. Different systems focus on various levels of coverage. It is important to understand how much of the space of potential visualizations is covered by the recommendation system [39].

The data-driven recommendation systems address these issues one way or another. However, the focus of such systems is typically a subset of these factors. In terms of quantifying interestingness and identifying visual encodings, the most recent research efforts have focused on learning human perception. In particular, understanding which visualization and transformation is good under which scenarios and using that learning to mine the interesting visualization. Moreover, such systems involve heavy interaction with the user in the learning phase, hence these system can come under the interaction-driven systems. However, focus of this interaction is off-line and the learning is taken as input to the system like the data, therefore, the recommendation process is dominated by the data-driven approach. The performance and capabilities of these systems can be improved by improving the dataset of examples, which is used to train the models.

DeepEye [10] is a ML based system with online and offline components, the off-line component trains two ML models. First model trains a decision tree to determine whether a given dataset and an associated visualization is good or bad. Second model is a neural-network to rank the visualizations. The on-line component generates all possible visualizations and uses the trained models to select top-k visualizations.

Data2Viz [11] is another learning based system that automatically generates visualizations using deep learning approach. Particularly, the visualizations generation is formulated as a language translation problem data specifications are mapped to visualization specifications in Vega-Lite. A multi-layered attention-based encoder-decoder network is trained with long short term memory units.

VizML [12] proposes another ML based approach for visualization recommendation that learns visualization design choices through a corpus of datasets and associated visualizations. Neural network classifiers are developed for five design choices prediction tasks. The work also benchmark with a test set through crowdsourcing.

Next, we review the systems that quantify interestingness in term of metrics directly related to data and not on any other sources such a learning from prior knowledge or domain expertise. QuickInsights [47] formulates insights based on three elements i.e., subject, type and interestingness to discover insights. Insight subject is the content of the insight such as the subset, the attributes and the transformation. It supports 12 types of insights such as correlation, outliers etc. To achieve efficient insights, QuickInsights proposes mechanism to prioritize insight evaluation tasks and smart query grouping to reduce the number of queries. The interestingness is measured as a combination of significance and impact score of insight. QuickInsights is released in Microsoft Power BI tool. In their previous work [49] they propose the concept of insight derived from aggregation result in multiple steps. The computation sharing and pruning based optimizations are also proposed.

Foresight [44] considers insight as a strong manifestation of a distributional property of data. It proposes insight classes such as, dispersion, skew, heavy tails, outliers etc., and the visualization types for each class. Foresight initially presents top-k instances in data based on different ranking. The user

Symbol	Description
D_B	Database
Q	Input query
T	Predicates for Input query
D_Q	Dataset selected by Q
V_i	i^{th} aggregate view
$V_i(D_Q)$	i^{th} target view
$V_i(D_B)$	i^{th} comparison view on D_B
$D(V_i)$	Deviation of aggregate view $V(i)$

Table 2.1: Summary of symbols

can then choose the interesting ones and explore further by issuing insight queries in specific format. It can handle large datasets by using approximate methods based on sketching and indexing.

VizDeck [7] generates all possible 1-D and 2-D visualizations ranked by statistical measures on a dashboard. The ordering of the visualizations can be adjusted by the proposed voting mechanism in which users can share, vote or save visualizations.

In data-driven recommendation, a huge number of visualizations are generated and comparisons are made to discover insights. As a result the probability of discovering spurious insights increase. This is known as multiple comparison problem (MCP) and is well studied in statistics. In the context of visualizations recommendations [50] investigate this problem and proposes methods to evaluate MCP by measuring the accuracy of insights with known ground truth labels.

Deviation-Based Data-Driven Recommendation Systems

As mentioned earlier, one of the main challenges in recommending data-driven visualizations is the quantification of interestingness which depends on a lot of factors. Recent work provides strong evidence that a deviation-based formulation of interestingness is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [6]. The underlying premise is that a visualization is likely to be interesting if it displays a large deviation from some reference (e.g., complete dataset, another dataset, historical data or the rest of the data).

Consider a database D_B, the user specifies an input query Q to select a subset D_Q of data from D_B . For instance, consider the following input query Q:

Q : SELECT * FROM D_B WHERE T;

In Q, T specifies a combination of predicates, which selects a portion of D_B for visual analysis. Typically, the reference dataset is either the complete database D_B. For deviation-based recommendation systems, a visualization shows comparison of the visual representation for target and reference dataset. For instance, a visual representation of Q is basically the process of generating an aggregate view V_i, composed of a target view $V_i(D_Q)$ on D_Q and a comparison view $V_i(D_B)$ on D_B, which is then plotted using some of the popular visualization methods (e.g., bar charts, scatter plots, etc.). Typically, for computing deviation a distance metric is defined as utility function which computes distance between $V_i(D_Q)$ and $V_i(D_B)$.

SeeDB [6] is one of the first system which recommends top-k aggregate views on user specified input query Q based on a deviation metric. It generates all possible aggregate views $V_i(D_Q)$ and $V_i(D_B)$ by generating GROUP BY queries on D_Q and the reference dataset D_B. It recommends the top-k visualizations based on distance between the corresponding $V_i(D_Q)$ and $V_i(D_B)$. It proposes multi-query optimization techniques to share computation among the candidate views. SeeDB also proposes a suit of approximate optimizations based on confidence-interval, top-k ranking and multi-arm bandits to prune low utility views. The user study in this work provides strong evidence that a deviation-based formulation of utility is able to provide analysts with interesting visualizations.

Ziggy [14] recommends a diverse set of attributes in which D_Q has unusual distribution compared to D_B. Ziggy focus on describing the general distribution of D_Q instead of aggregate views and it builds the views one by one, in a greedy manner. It uses a hybrid utility function consisting of dissimilarity and diversity value of views. The dissimilarity metric is a combination of several simple indicators of dissimilarity such as, difference between means etc. It also evaluates the statistical robustness of the views using asymptotic bounds on each component of utility and aggregation of their confidence scores.

TopKAttr [15] finds top-k attributes whose distributions in the D_Q and D_B deviate most from each other. Specifically, the user specifies the Q, k and deviation-based utility metric. The system considers all views $V_i(D_Q)$ and corresponding $V_i(D_B)$ using all attributes and `COUNT` aggregate function. The system recommends top-k bar chart *histograms*.The deviation is measured by normalized l_1 (total variation or earth movers distance) or l_2 (Euclidean distance). A sampling based solution is proposed that is guaranteed to return the correct top-k attributes with high probability. Novel analytic techniques are also proposed to derive confidence intervals for deviation based utility functions. The proposed solution is highly scalable as it can produce 25x speedup with near-zero error in the answer.

Manually exploring all possible subsets of data for insights can be tedious and inefficient. VISPILOT [51] identifies a network of visualizations that convey key insights in the dataset based on deviation between $V_i(D_Q)$ and $V_i(D_B)$. VISPILOT constructs a lattice of visualizations, where the visualization from D_B are at the top and the next level has visualizations from subsets having single predicate and so on. The lattice is traversed from top to bottom. VIZPILOT introduces a concept of informativeness in terms of measure of similarity between a visualization and its parent. The utility of a visualization is the distance between the visualization and its most informative parent.

[52, 53] is another work that automatically explores the subsets of data. It considers all possible subsets of data specified by a single predicate, where predicate is a categorical attribute. The comparison views $V_i(D_B)$ belong to the overall database D_B. To efficiently navigate the search space of subsets of data they employ the multi query optimization and confidence interval strategies.

Some recent research efforts in deviation-based data-driven recommendation systems have focused on false discoveries. For instance, VizRec [48] is framework that quantifies the statistical significance of recommended visualizations to improve performance. VizRec recommends a visualization $V_i(D_Q)$ for a reference $V_i(D_B)$ if their corresponding histograms are statistically different with respect to the true underlying distribution of D_B. The false discoveries are controlled by classical statistical testing

System	**Visualization Type**	**Distance Metric**	**Numeric Dimensions**	**User Preferences**	**Scalability**	**False Discoveries**	**Coverage**
SeeDB [6]l	Bar Charts	Earth Movers	x	x	v	x	x
Ziggy [14]	Scatter Plot	Combination of Statistical Measures	x	x	x	v	x
TopKAttr [15]	Histograms	Euclidean	x	x	v	x	x
VizRec [48]	Histograms	Chebyschev	x	x	x	v	x
VISPILOT [51]	Bar Charts	Euclidean	x	x	x	v	v
Efficient data slice search [52]	Bar Charts	Euclidean	x	x	v	x	v
View-360	**Bar Charts**	**Euclidean**	**v**	**v**	**v**	**v**	**v**

Table 2.2: Deviation-Based Data-Driven Visualization Recommendation Systems

and by application of Vapnik Chervonenkis dimension method. QUDE [54, 55] is a system, focused on automatically controlling risk factors such as, multiple hypothesis testing and simpson's paradox during the data exploration process.

View-360

The current data-driven visualization recommendation systems are still primitive. Particularly, in deviation-based data-driven recommendation systems, the existing systems fails to capture the requirements of numerical dimensions. Moreover, most of the work in this area lacks coverage in terms of making recommendations from complete search space i.e., considers only a subset of data for visualization recommendation.

View-360 is the prototype system we developed that combines all of the schemes proposed in this thesis. The table 2.2 compares the features supported by other systems and our proposed end to end prototype View-360. As seen in the table, the default distance metric we support is euclidean distance, however, any other distance metric can also be plugged in if required. We also support recommendation on numerical dimension attributes. View-360 includes query refinement based schemes that provide maximum coverage by exploring all possible subsets of data. We also include user preferences in a way that it would fit for different categories of users. Particularly, the user sets various exploration parameters in the beginning or she can choose to use default settings. View-360 also integrates hypothesis testing to exclude false discoveries from the recommendations. In this thesis we have primarily proposed pruning and approximation based schemes to make the solution scalable for large datsets.

Chapter 3

Efficient Binned View Recommendation

3.1 Introduction

Recommending data visualizations that reveal new and valuable insights is a challenging problem, which has been the focus of many research approaches (e.g., [6–9, 17]). The main idea underlying those approaches is to automatically generate all possible aggregate views of data, and recommend the *top-k* views that result in interesting visualization, where the interestingness of a visualization is quantified according to some utility function. Recent work provides strong evidence that a deviation-based formulation of utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [6, 17]. The underlying premise is that a visualizations that results in a higher deviation is expected to reveal some interesting insights that are very particular to the analyzed dataset [6, 17]. While the deviation-based notion of utility has been shown to be effective in recommending views with categorical dimensional attributes, in this work we argue that it falls short in capturing the requirements of numerical dimensions. Particularly, in the presence of such numerical dimensions, *binned aggregation* is typically required to group the numerical values along a dimension into adjacent intervals [18, 19]. Given the large number of options for binning a numerical dimension, it is expected that different binning configuration will result in different deviations, and in turn, different levels of interestingness from the analyst point of view. For instance, in a view with small number of bins, interesting insights are expected to remain hidden under a smooth and coarse visual representation. Meanwhile, in a view that contains a large number of bins, insights might go unnoticed in a cluttered or sparse visualization. To illustrate the impact of binning on numerical dimensions, consider the following example:

Example 4. *Consider a data analyst trying to gain insights into the special factors that led the Golden State Warriors (GSW) basketball team to win the 2015 NBA championship. Consequently, the analyst uses the 2015 NBA players statistics database [77] to compare the GSW team to the other teams in the league. Particularly, the analyst poses a query:*

```
Q: SELECT * FROM players WHERE team=GSW,
```

which returns the data for all the players on the GSW team. The data include different dimensions

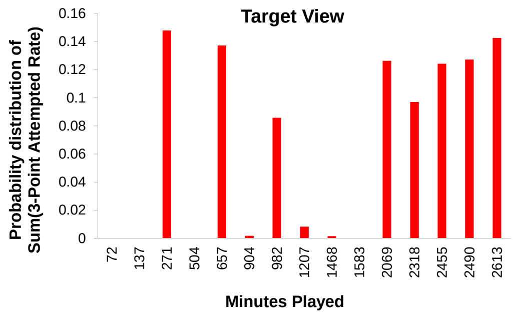


Figure 3.1: View on players of the GSW team (target view)

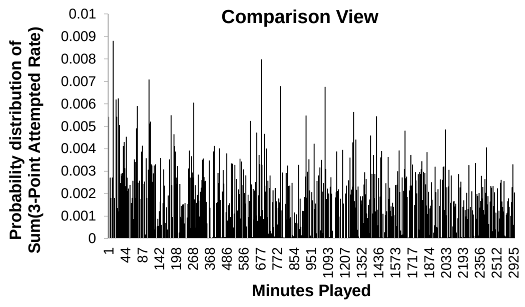


Figure 3.2: View on all players in the 2015 NBA (comparison view)

(e.g., age, number of games played, minutes played, etc.), and different measures (e.g., player efficiency rating, 3-point attempt rate, etc.). To recommend interesting bar chart visualizations, different SQL aggregate functions are applied on the views resulting from all the possible pairwise combinations of dimensions and measures, then the most interesting views are presented to the analyst. Figure 3.1 shows one particular view defined on the dimension `minutes played (MP)` *and the measure* `3-point attempt rate (3PAr)`. *Such view is equivalent to:*

```
V: SELECT MP, SUM (3PAr) FROM players WHERE team=GSW GROUP BY MP
```

Meanwhile, generating the same view of the entire database of all players (i.e., without the `WHERE team=GSW` *clause), results in the visualization shown in Figure 3.2. At first glance, comparing the two views fails to reveal any insights about the GSW team. However, binning the two views, as shown in Figure 3.3, reveals some very interesting observation. Particularly, Figure 3.3 shows that for all NBA players, the* `3PAr` *decreases as they play more games. The intuitive explanation is that the fatigue incurred from playing more games can affect their fitness and reduce their* `3PAr`. *However, for the GSW players, that pattern significantly deviates from that general pattern. As Figure 3.3 shows, the GSW players who spent more time on the field still achieve very high* `3PAr`. *In fact, their* `3PAr` *is almost 4 times that of the other players. Clearly, that observation reflects the fitness and consistency of the GSW players, which might distinguish them from other players in the league, and can shed some light into understanding their championship win.*

Choosing the right binning is essential in the process of extracting insights from the data, whether that process is performed manually or analytically. On the one hand, a good binning allows to reduce both the clutter and sparsity in the generated visualizations, which makes them easy to use by the analyst to manually extract insights [56, 57]. On the other hand, a good binning also allows to group similar data together, so that the special features of each group is aggregated and emphasized, which

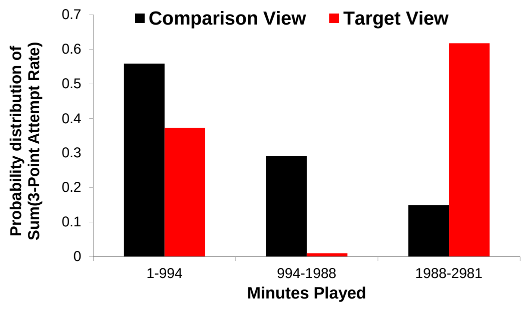


Figure 3.3: Binned target view (i.e., GSW team) and comparison view (i.e., all NBA teams)

in turn allows quantitative metrics, such as deviation, to capture the interesting patterns exhibited by those features. We note, however, that choosing the right binning for each visualization is a non-trivial task. The benefits, as well as the challenges, of binning numerical dimensions are well-recognized in the literature, especially in the context of histogram construction for the purpose of selectivity estimation and query optimization (e.g., [18, 19, 58]). Such histograms provide a concise summary of the underlying data distribution of an attribute, where the accuracy of that summarization is dependent on the employed binning strategy. Similarly, in bar chart visualizations, which is the focus of this thesis, the overall utility of a visualization is dependent on the underlying binning. Consequently, the applicability of the simple deviation-based notion of utility becomes very limited in the presence of numerical dimension attributes.

To address such limitation, we introduce a novel hybrid multi-objective utility function, which captures the impact of numerical dimension attributes in terms of generating visualizations that are: 1) interesting, 2) usable, and 3) accurate. Clearly, combining these often conflicting objectives dramatically expands the search space of possible visualizations (i.e., aggregate views). Moreover, it significantly increases the processing time incurred to asses the overall utility of each view, which is assembled from the utility values of each of the three objectives listed above.

Accordingly, we propose a suite of novel search algorithms, which are particularly optimized to leverage the specific features of the view recommendation problem. The main idea underlying our first scheme *Multi-Objective View Recommendation for Data Exploration* (**MuVE**) is to use an incremental evaluation of the multi-objective utility function, where different objectives are computed progressively. Our results in [1] show that MuVE is able to prune a large number of unnecessary views, and in turn reduces the overall processing time for recommending the top-k views. However, that achieved pruning power is highly dependent on the order in which those views are presented to MuVE and might often limit its performance gains. To address that limitation, we propose our second scheme *upper MuVE* (**uMuVE**), in which the goal is to provide a flexible navigation of the search space so that high-utility views are discovered earlier. Particularly, uMuVE is based on setting upper bounds on the utility of each possible view, which is then exploited to effectively guide the search process by means of interleaving the evaluation of the different objectives offered by the different views. Due to that interleaved processing, at any point of time, uMuVE would typically have multiple views under consideration, which requires significant amount of memory for storing their data. This motivated us to propose our third scheme *Memory-aware uMuVE* (**MuMuVE**), which aims to provide a pruning power close to that of uMuVE under some memory usage constraints.

The most expensive operation while computing the utility of views is the time spent in executing the queries related to the views. To reduce the cost of this particular operation, a novel technique **mView** is proposed, which instead of answering each query related to a view from scratch, *reuses results* from the already executed queries. In summary, this is done by materializing views and answering queries from the materialized views instead of the base table. The idea of materializing views for reducing the query-processing time is well studied in the literature [59–61] and has proven significant relevance to a wide variety of domains, such as query optimization, data integration, mobile computing and data warehouse design [60, 61]. However due to prohibitively large number of views, the blind application of materialization may result in even further degradation of the cost [59]. Substantial amount of work has already been done to select an appropriate set of views to materialize that minimize the total query response time and the cost of maintaining the selected views, given a limited amount of resource, e.g., materialization time, storage space etc. [62]. In this work our proposed technique **mView** first defines a cost benefit model to decide which views are the best to reuse. Later, in an optimal order it materializes the best set of views, which reduce the overall cost of the solution.

We also included set of experiments to demonstrate efficiency of all of our proposed view recommendation techniques.

3.2 Related Work

Alongside the commercial tools such as tableau, Qlik etc., several research efforts have been directed towards providing automation features to the visualization process. Show Me [63] is an add-on for commercial visualization tool Tableau, it provides automatic presentation by means of additional user interface commands and defaults. It includes a restrictive feature of ranking views according to its graphical presentation type. The user still has to decide which dimensions and measures to visualize. Rank-by-Feature Framework [45] computes statistical summaries and ranking for histograms and scatter-plots. While visualizations can be ranked by various features, the user still has to select a ranking criterion, and then all possible projections are ranked by that criterion. Profiler [8] detects anomalies and recommends visualizations based on mutual information metric. Hence, it is specifically designed to highlight data quality issues, but exploring data for interesting view is beyond the scope of that work. VizDeck [7] generates all possible 2-D visualizations on a dashboard and allows users to reorder, share or permanently store those visualizations. It also recommend views, based on a visualization quality model using statistical features of the dataset such as VizDeck lacks the deviation based ranking and it does not scale for high dimensional large datasets. The ziggy approach [9, 14] introduces a multi-view subset characterization approach based on the idea of selecting tuples that differ from the rest of the database. It recommends sets of columns on which user selected data has an unusual distribution from the rest of the database. Specifically, it performs tuple based ranking, while our work focuses on binned aggregate views. As mentioned earlier, our work recommends visualizations based on the deviation between two datasets, as in SeeDB [6, 17]. However, while

SeeDB effectively recommend views for categorical attributes, it lacks the necessary techniques for handling numerical attributes, which is focus of our work.

3.3 Preliminaries

3.3.1 View Recommendation

The process of visual data exploration is typically initiated by an analyst specifying a query Q on a database D_B. The result of Q, denoted as D_Q, represents a subset of the database D_B to be visually analyzed. For instance, consider the following query Q:

Q: SELECT * FROM D_B WHERE T;

In Q, T specifies a combination of predicates, which selects a portion of D_B for visual analysis (e.g., `team = GSW`). A visual representation of Q is basically the process of generating an aggregate view V of its result (i.e., D_Q), which is then plotted using some of the popular visualization methods (e.g., bar charts, scatter plots, etc.). Similar to traditional OLAP systems and recent data visualization platforms [6–8, 17, 64], our model is based on a multi-dimensional database D_B, consisting of a set of dimension attributes $\mathbb{A}$ and a set of measure attributes $\mathbb{M}$. Additionally, $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$, such as SUM, COUNT, AVG, STD, VAR, MIN and MAX. Hence, an aggregate view V_i over D_Q is represented by a tuple (A, M, F) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, and $F \in \mathbb{F}$. That is, D_Q is grouped by dimension attribute A and aggregated by function F on measure attribute M (all symbols are summarized in Table 4.1). As in [6], we consider aggregate views that perform a single-attribute group-by and aggregation on D_Q. A possible view V_i of the example query Q above would be expressed as:

V_i: SELECT A, F(M) FROM D_B WHERE T
GROUP BY A;

where the GROUP BY clause specifies the dimension A for aggregation, and $F(M)$ specifies both the aggregated measure M and the aggregate function F.

Typically, a data analyst is keen to find visualizations that reveal some interesting insights about the analyzed data D_Q. However, the complexity of this task stems from: 1) the large number of possible visualizations, and 2) the interestingness of a visualization is rather subjective. Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on some objective, well-defined quantitative metrics (e.g., [6–8, 17]). Among those metrics, recent case studies have shown that a *deviation-based* metric is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [6].

In particular, the deviation-based metric measures the distance between $V_i(D_Q)$ and $V_i(D_B)$. That is, it measures the deviation between the aggregate view V_i generated from the subset data D_Q vs. that generated from the entire database D_B, where $V_i(D_Q)$ is denoted as *target* view, whereas $V_i(D_B)$ is denoted as *comparison* view. The premise underlying the deviation-based metric is that a view V_i that results in a higher deviation is expected to reveal some interesting insights that are very particular to the subset D_Q and distinguish it from the general patterns in D_B. To ensure that all views have the same scale, each target view $V_i(D_Q)$ is normalized into a *probability distribution* $P[V_i(D_Q)]$ and each comparison view into $P[V_i(D_B)]$. This visualization is not useful for any insight, the target and comparison views are plotted on the same graph but have different scales, as the number of tuples in target view are fewer than the comparison view.

Consider an aggregate view $V = (A, M, F)$. The result of that view can be represented as the set of tuples: $<(a_1, g_1), (a_2, g_2), ..., (a_t, g_t)>$, where t is the number of distinct values (i.e., groups) in attribute A, a_i is the i-th group in attribute A, and g_i is the aggregated value $F(M)$ for the group a_i. Hence, V is normalized by the sum of aggregate values $G = \sum_{p=1}^{t} g_p$, resulting in the probability distribution $P[V] =< \frac{g_1}{G}, \frac{g_2}{G}, ..., \frac{g_t}{G} >$. For instance, for the comparison view shown in Figure 3.3, Table 3.1 illustrates the groups (Minutes played), aggregate values (Sum 3-PAR) and the computation of its probability distribution.

For a view V_i, given the probability distributions of its target and comparison views, the deviation $D(V_i)$ is defined as the distance between those probability distributions. Formally, for a given distance function *dist* (e.g., Euclidean distance, Earth Mover's distance, K-L divergence, etc.), $D(V_i)$ is defined as:

$$D(V_i) = dist(P[V_i(D_Q)], P[V_i(D_B)]) \tag{3.1}$$

Consequently, the deviation $D(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) [6]. Hence, the number of possible views to be constructed is $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$, which is clearly inefficient for a large multi-dimensional dataset. Thus, several techniques have been proposed for optimizing the processing time incurred in recommending visualizations, which are orthogonal to the optimizations proposed in this work to address the impact of numerical dimensions, which is described next.

3.3.2 Numerical Dimensions

In this thesis, we mainly focus on the problem of recommending visualizations in the presence of numerical dimension attributes. While numerical dimension attributes (e.g., age, height, etc.) are abundant in scientific and commercial databases, current visualization recommendation techniques tend to mostly overlook such numerical dimensions, and rather focus on the categorical ones. In the presence of numerical dimensions, *binned aggregation* is typically required so that to group the numerical values along a dimension into adjacent intervals over the range of values covered by that dimension [18, 19]. Accordingly, binning of numerical dimensions poses several non-trivial challenges in terms of recommending visualizations that are not only interesting, but also *accurate* and *usable*. Particularly, in addition to being interesting (i.e., highly deviated from the general data D_B), recommended visualizations are expected to be accurate (i.e., minimize the amount of *error* between the aggregated view V_i and its corresponding dataset D_Q) and usable (i.e., minimize the amount of *clutter* in view V_i). For instance, while the target and comparison views shown in Figures 3.1 and 3.2

Comparison View

Minutes Played	Sum (3-PAR)	Pdf Sum(3-PAR)
$a_1 : 1 - 994$	$g_1 : 103.32$	$\frac{g_1}{G} = \frac{103.32}{184.87} = 0.56$
$a_2 : 994 - 1988$	$g_2 : 53.97$	$\frac{g_2}{G} = \frac{53.97}{184.87} = 0.29$
$a_3 : 1988 - 2981$	$g_3 : 27.58$	$\frac{g_3}{G} = \frac{27.58}{184.87} = 0.15$
Sum of aggregate values $(G) = 184.87$		

Table 3.1: Computing the probability distribution of the comparison view shown in Figure 3.3.

are highly accurate (no binning applied), they are also barely usable because of high clutter or high sparsity, which translates into missing out on revealing interesting insights.

As mentioned earlier, the benefits, as well as the challenges, of binning numerical dimensions are well-recognized in the literature, especially in the context of histogram construction (e.g., [18, 19, 58]), anomaly detection (e.g., [65]), and data visualization (e.g., [8, 66]). For instance, binning (also know as bucketing) is an essential step in constructing histograms over numerical attributes for the purpose of selectivity estimation and query optimization. While a histogram that is based on a small number of bins provides a high degree of compression, its accuracy can be quite poor. To the contrary, adding more bins to a histogram, or equivalently decreasing its bin width, has been shown to increase the accuracy of a histogram at the expense of increasing the costs incurred in processing, maintaining, and storing those additional bins. Similarly, in bar chart visualizations, which is the focus of this thesis, too few bins result in loss of information and compromise the accuracy of visualization, while too many bins result in a cluttered low quality visualization.

Deciding the optimal bin width for histograms has been intensively studied in the statistics literature, where several *model-based* approaches have been proposed [18]. In contrast, the database literature mostly takes a *model-free* approach, considering the dataset currently stored in the database as the only data of interest. (We refer the reader to [18, 19], for comprehensive surveys on that topic.) In this work, we adopt the same approach and expand on existing model-free methods, as discussed next.

3.3.3 Binned Views

To enable the incorporation and recommendation of visualizations that are based on continuous numerical dimensions, in this work we introduce the notion of a *binned view*. A binned view $V_{i,b}$ simply extends the basic definition of a view to specify the applied binning aggregation. Specifically, given a view V_i represented by a tuple (A, M, F), where $A \in \mathbb{A}$, $M \in \mathbb{M}$, $F \in \mathbb{F}$, and A is a continuous numerical dimension with values in the range $L = [L_{min} - L_{max}]$, then a binned view $V_{i,b}$ is defined as:

Definition 5. Binned View: *Given a view V_i and a bin width of w, a binned view $V_{i,b}$ is a representation of view V_i, in which the numerical dimension A is partitioned into a number of b equi-width non-overlapping bins, each of width w, where $0 < w \leq L$, and accordingly, $1 \leq b \leq \frac{L}{w}$.*

For example, Figure 3.3 shows a binned view, in which the number of bins $b = 3$ and the bin width $w = 994$.

We note that our definition of a binned view resembles that of an equi-width histogram in the sense that a bin size w is uniform across all bins. While other non-uniform histograms representations (e.g., equi-depth and V-optimal) often provide higher accuracy when applied for selectivity estimation, they are clearly not suitable for standard bar chart visualizations.

Given our binned view definition, a possible binned bar chart representation of query Q is expressed as:

$V_{i,b}$: `SELECT A, F(M) FROM` D_B `WHERE T`

```
GROUP BY A
NUMBER OF BINS b
```

The deviation provided by a binned view $V_{i,b}$ is computed similar to that in Eq. 4.1. In particular, the comparison view is binned using a certain number of bins b and normalized into a probability distribution $P[V_{i,b}(D_B)]$. Similarly, the target view is binned using the same b and normalized into $P[V_{i,b}(D_Q)]$. Then the deviation $D(V_{i,b})$ is calculated as:

$$D(V_{i,b}) = dist(P[V_{i,b}(D_Q)], P[V_{i,b}(D_B)]) \tag{3.2}$$

Clearly, the deviation $D(V_{i,b})$ is bounded by a maximum value D_M. That value D_M is achieved when for each group a_i at least one of the $\frac{g_i}{G}$ from $P[V_{i,b}(D_B)]$ and $P[V_{i,b}(D_Q)]$ is zero. Therefore, to normalize the deviation, $D(V_{i,b})$ is divided by D_M.

Clearly, for a binned view such as view $V_{i,b}$ defined above, its *usability* depends on the visual quality i.e., how clearly the visualization reveals the structure within the data. Inversely, clutter is defined by the crowdedness that obscure the structure within the visualization [57]. A number of metrics have been proposed to measure clutter for various types of visualizations, such as the number of data points displayed, data density (number of data points/number of pixels), data to ink ratio and lie factor [56, 57]. All of these metrics basically quantify the amount of content displayed on screen as a measure of clutter. Similarly, for bar chart visualizations, clutter occurs due to the large number of bins in a visualization. Consequently, we simply define usability as the inverse of clutter, which is captured as follows:

$$S(V_{i,b}) = \frac{1}{b} = \frac{w}{L} \tag{3.3}$$

where b is the number of bins, w is the width of each bin and L is the range of the dimension attribute. $S(V_{i,b})$ is in the range $[0,1]$, such that $S(V_{i,b}) = 1$ indicates highest quality.

Furthermore, a binned view $V_{i,b}$ is obviously a summarized approximation of the corresponding non-binned view V_i. Thus, it is essential to measure the (in)accuracy provided by $V_{i,b}$. To achieve this, consider a non-binned view V_i, which is defined as (A, M, F). Further, and without loss of generality, assume A is an ordered integer attribute. As described in the previous section, the result of that view can be represented as the set of tuples: $<(a_1, g_1), (a_2, g_2), ..., (a_j, g_i), ..., (a_t, g_t)>$, where t is the number of distinct values (i.e., groups) in attribute A. A binned view $V_{i,b}$ provides a concise approximate representation of V_i based on partitioning the ordered attribute A into b bins. Particularly, each bin I_x consists of a start and end point, $I_x = (s_x, e_x)$, and a value $\hat{g}_x$, which represents the aggregated value of the measure M over all the values of dimension A in the range of I_x. That is, $V_{i,b} =< (I_1, \hat{g}_1), (I_2, \hat{g}_2), ..., (I_x, \hat{g}_x), ..., (I_b, \hat{g}_b) >$, where $b \ll t$.

This data reduction implies approximation errors in the estimation of the original non-binned aggregate values, where the error incurred by that approximation increases with decreasing the number of bins b. There are number of metrics that have been used for measuring that kind of (in)accuracy such as Sum Squared Error, Sum-Absolute-Error, Sum-Absolute-Relative-Error and Maximum-Absolute Relative-Error [67]. In this work, we adopt Sum Squared Error (SSE) metric, which has also been widely employed by the database community to measure the accuracy of frequency histograms for the purpose of query optimization (e.g., [58,68]). Applying SSE metric for general aggregate views is straightforward. In particular, the aggregate measure corresponding to any dimension value in the contiguous range $s_x, s_x + 1, ..., e_x$ is approximated using a single representative value g'_x, which is computed as $\frac{\hat{g}_x}{n_x}$, where $n_x = e_x - s_x + 1$ (i.e., the number of distinct values in bin b_x). Accordingly, each $g_j \in I_x$ is estimated as $g'_j = g'_x$. Hence, the SSE provided by $V_{i,b}$, denoted as $E(V_{i,b})$, is computed as $E(V_{i,b}) = \sum_{p=1}^{t} (g_p - g'_p)^2$ and the relative SSE is computed as $R(V_{i,b}) = \sum_{p=1}^{t} \frac{(g_p - g'_p)^2}{g_p^2}$. Accordingly, the accuracy of a view $V_{i,b}$ is simply computed as:

$$A(V_{i,b}) = 1 - \frac{R(V_{i,b})}{t} \tag{3.4}$$

The computed $A(V_{i,b})$ is in the range $[0,1]$, such that $A(V_{i,b}) = 1$ indicates maximum accuracy (i.e., zero error).

Clearly, incorporating the different metrics listed above further complicates the problem of finding the top-k recommended visualizations. This is mainly due to the different binning options, which in turn leads to an increase in the number of candidate visualizations.

3.4 Multi Objective View Recommendation

3.4.1 Problem Definition

In a nutshell, the goal of this work is to recommend the *top-k* bar chart visualizations of the results of query Q according to some utility function. When all dimension attributes are categorical, such goal simply boils down to recommending the *top-k* interesting views based on the deviation metric [6,17], as described in Section 4.3.1. However, that simple notion of utility falls short in capturing the impact of numerical dimensions. In particular, the presence of numerical dimensions introduces additional factors that impact the utility gained from a recommended view. In our proposed schemes, we employ a novel hybrid multi-objective utility function, which integrates such factors, namely:

1. *Interestingness:* Is the ability of a view to reveal some interesting insights about the data, which is measured using the deviation-based metric $D(V_{i,b})$ (Eq. 3.2).

2. *Usability:* Is the quality of the visualization in terms of providing the analyst with an understandable uncluttered representation, which is quantified via the relative bin width metric $S(V_{i,b})$ (Eq. 3.3).

3. *Accuracy:* Is the ability of the view to accurately capture the characteristics of the analyzed data, which is measured in terms of the accuracy metric $A(V_{i,b})$ (Eq. 3.4).

Notice that the different factors listed above are often at odds with each other. For instance, a view that contains a large number of bins can provide high accuracy, at the expense of being cluttered and difficult to understand by an analyst. To the contrary, using a small number of bins leads to a very smooth and coarse representation of the analyzed data, which can hide its particular and interesting characteristics. To capture those conflicting factors, MuVE employs a weighted sum multi-objective utility function, which is defined as follows:

$$U(V_{i,b}) = \alpha_D \times D(V_{i,b}) + \alpha_A \times A(V_{i,b}) + \alpha_S \times S(V_{i,b}) \tag{3.5}$$

where $D(V_{i,b})$ is the normalized deviation of binned view $V_{i,b}$ from the overall data, $A(V_{i,b})$ is the accuracy of $V_{i,b}$, and $S(V_{i,b})$ is the usability of $V_{i,b}$.

Parameters α_D, α_A and α_S specify the weights assigned to each objective in our hybrid utility function, such that $\alpha_D + \alpha_A + \alpha_S = 1$. Those weights can be user-defined so that to reflect the user's preference between the three aspects of utility. Also, notice that all objectives are normalized in the range $[0,1]$. Accordingly, the overall multi-objective utility function takes value in the same range (i.e., $[0,1]$), where the goal is to maximize that overall utility. Such goal is formulated as follows:

Definition 6. Multi-Objective View Recommendation: *Given a user-specified query Q on a database D_B, a multi-objective utility function U, and a positive integer k, find the k aggregate binned views over D_Q, which have the highest utility values.*

In summary, we posit that a view is of high utility, if it shows a unique pattern that is based on accurate data and can be visually identified and appreciated by the user. For instance, referring back to our motivating Example 4 in Chapter 1, while Figure 3.1 shows a non-binned view (i.e., accuracy of 1.0), the deviation provided by that view is only 0.17, and its usability is ~0. Meanwhile, Figure 3.3 shows a binned version of the same view obtained at $\alpha_A = 0.2, \alpha_D = 0.6, \alpha_S = 0.2$, which results in deviation=0.29, usability=0.33, and accuracy=0.30. That increase in both deviation and usability, allowed that particular view to come first on the view recommendation list (i.e., *top-1*), and enabled for an insightful visualization.

View Processing Cost

Recall that in the absence of numerical dimensions, the number of candidate views N to be constructed is equal to $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. In particular, $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ queries are posed on the data subset D_Q to create the set of target views, and another $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ queries are posed on the entire database D_B to create the corresponding set of comparison views. In addition, a total of $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ distance computations are needed to calculate the deviation between each pair of target and comparison views. For each candidate non-binned view V_i over a numerical dimension A_j, the number of target and comparison binned views is equal to: $2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$, where B_j is the maximum number of possible bins that can be applied on dimension A_j (i.e., number of binning choices). Hence, in the presence of $|\mathbb{A}|$ numerical dimensions, the total number of binned views grows to N_B, which is simply calculated as:

$$N_B = \sum_{j=1}^{|\mathbb{A}|} 2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j \tag{3.6}$$

Furthermore, for each pair of target and comparison binned views, the three metrics/objectives listed above are to be evaluated. Evaluating those metrics incurs the following processing costs:

1. *Query Execution Time*: Is the time required to process the raw data to generate the candidate target and comparison binned views, where the cost for generating the target view is denoted as $C_t(V_{i,b})$, and that for generating the comparison view is denoted as $C_c(V_{i,b})$.

2. *Deviation Computation Time*: Is the time required to measure the deviation between the target and comparison binned views, and is denoted as: $C_d(V_{i,b})$. Notice that this time depends on the employed distance function $dist$.

3. *Accuracy Evaluation Time*: Is the time required to measure the accuracy of the binned target view in representing the underlying data distribution and is denoted as $C_a(V_{i,b})$.

Putting it together, the total cost incurred in processing a candidate view V_i is expressed as:

$$C(V_i) = \sum_{b=1}^{B} C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b}) + C_a(V_{i,b}) \tag{3.7}$$

Hence, the total cost incurred in processing all candidate binned views is expressed as:

$$C = \sum_{i=1}^{N_B} C(V_i) \tag{3.8}$$

3.4.2 Search Strategy Overview

In section 3.4.3- 3.4.7, we present search strategies for finding the top-k binned views for recommendation. For clarity of presentation, we break down a search strategy into two integral components, namely: 1) *Horizontal* Search, and 2) *Vertical* Search, as shown in Figure 3.4. At a high level, the objective of horizontal search is to find the optimal binning for a given non-binned view, whereas the objective of vertical search is to find the top-k binned views with the highest utility values. In Section 3.4.3- 3.4.6, we present different strategies for horizontal search, whereas in Section 3.4.7, we expand on those strategies to enable and integrate vertical search.

The total number of binned target and comparison views are N_B as defined in Eq. 3.6. Evaluating the utility of each pair of those target and comparison binned views requires a total processing time $C(V_i)$, which includes the times needed for query execution, deviation computation, and accuracy

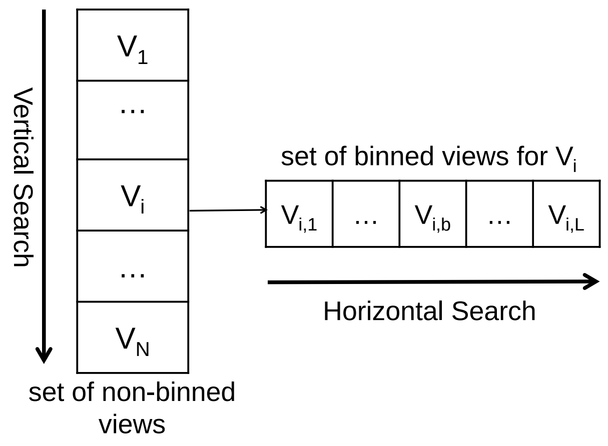


Figure 3.4: Horizontal and Vertical Searches for recommending top-k visualizations

evaluation. The large number of possible binned views, together with the complexity of evaluating the utility function, makes the problem of finding the optimal binning for a certain view V_i highly challenging. In the following, we present the optimal baseline scheme namely Linear Search together with our proposed schemes; i) Multi-Objective View Recommendation for Data Exploration (MuVE), ii) Upper bound based MuVE (uMuVE), and iii) Memory-aware uMuVE (MuMuVE). We also present a baseline approximation schemes namely local search and our proposed approximation schemes.

3.4.3 Baseline Schemes

Linear Search

Linear search is basically an exhaustive brute force strategy, which serves as a baseline for our evaluation. In this strategy, given a certain candidate non-binned view V_i, all its corresponding binned views are generated and the overall utility of each of those views is evaluated. Particularly, a non-binned view $V_i = (A, M, F)$ is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, ..., V_{i,b}, ..., V_{i,L}\}$, where b is the number of bins, and L is the range of the continuous numerical dimension A. Consequently, the value of b that results in the highest utility is selected as the binning option for view V_i resulting in the binned view $V_{i,opt}$.

Local Search

Local search is a meta heuristic method that is widely used in solving optimization problems. In general, a local search algorithm starts out with an initial solution and then attempts to find a better solution in the neighborhood of that initial one. In this work, we adopt dynamic *Hill Climbing (HC)*, with halving search as another baseline method [69]. Specifically, for the set of binned views $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,L}\}$, HC initially starts at some random number of bins b, where $1 \leq b \leq L$, and a step $s = L$. In each iteration of HC, it considers two alternative settings for the number of bins: $b - s$, and $b + s$, then moves to the one which provides maximum utility. When HC cannot find a move that increases the utility, then s is halved. This halving continues until $s < 1$. Despite of being susceptible to hitting local maxima, HC is expected to provide significant reductions in processing costs compared to the linear search methods.

3.4.4 The MuVE Scheme

Similar to the linear search described above, for a given non-binned view $V_i = (A, M, F)$, our MuVE scheme considers the set of all its possible binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,b}, ..., V_{i,L}\}$. Unlike linear search, however, MuVE reduces the computational costs incurred in processing that set by means of: 1) pruning unnecessary views, and 2) pruning unnecessary utility evaluations.

To easily understand MuVE, notice that our problem of searching the space and ranking binned views according to our multi-objective utility function Eq. 3.5 is similar to *Top-K* preference query processing. Particularly, for a view $V_{i,b}$, the three objectives $D(V_{i,b}), A(V_{i,b}), S(V_{i,b})$ can be perceived as the preference query over 3-dimensions. However, efficient algorithms for preference query processing (e.g., [70, 71]), are not directly applicable to our problem because: 1) For any view $V_{i,b}$ the values of $D(V_{i,b})$ and $A(V_{i,b})$ are not physically stored and they are computed on demand based on the binning choice b, and 2) The size of the view search space $\mathbb{V}_i$ is prohibitively large and potentially infinite. To address these limitations, MuVE adapts and extends algorithms for Top-K query processing towards efficiently and effectively solving the multi-objective view recommendation problem.

Before describing MuVE in details, we first outline a baseline solution based on simple extensions to the *Threshold Algorithm (TA)* [71]. Conceptually, to adapt the well-know TA to the view recommendation model, each possible binned view $V_{i,b}$ is considered as an object with three partial scores: 1) deviation $\alpha_D D(V_{i,b})$, 2) Accuracy $\alpha_A A(V_{i,b})$, and 3) Usability $\alpha_S S(V_{i,b})$. Those partial scores are maintained in three separate lists: 1) D-list, 2) A-list, and 3) S-list, which are sorted in descending order of each score. Under the classical TA algorithm, the three lists are traversed sequentially in a round-robin fashion. While traversing, the binned view with the maximum utility seen so far is maintained along with its utility. An upper bound on the utility (i.e., threshold) is computed by applying the utility function to the partial components of the last seen view in the three different lists. TA terminates when the maximum utility seen so far is above that threshold or when the lists are traversed to completeness.

Clearly, such straightforward conceptual implementation of TA is infeasible to our problem due to the limitations mentioned before. However, recall that the usability objective S is based on the number of bins in a view and is calculated as $S(V_{i,b}) = \frac{w}{L} = \frac{1}{b}$. Hence, out of the three lists mentioned above, a sorted list S can easily be generated at a negligible processing cost. In particular, given a view V_i over a numerical dimension A of range L, MuVE progressively populates the S-list with the values $\alpha_S S(V_{i,1}), \alpha_S S(V_{i,2}), ..., \alpha_S S(V_{i,L})$, which are the values of the usability objective sorted in decreasing order.

One possible approach for populating the D-list and A-list is to first generate the S-list and then compute the corresponding $D(V_{i,b})$ and $A(V_{i,b})$ values for each view $V_{i,b}$. Those values are then sorted in descending order and the TA algorithm is directly applied on all three lists. Clearly, that approach has the major drawback of incurring the cost for computing the deviation and accuracy of all the possible binned views. Instead, we leverage the particular *Sorted-Random (SR)* model of the Top-K problem to minimize the number of those expensive estimation probes.

The SR model is particularly useful in the context of web-accessible external databases, in which one or more of the lists involved in an objective function can only be accessed in random and at a high-cost [70–72]. Hence, in that model, the sorted list basically provides an initial set of candidates, whereas random lists (i.e., R) are probed on demand to get the remaining partial values of the objective function. In our model, the S-list already provides that sorted sequential access, whereas the D-list and A-list are clearly external lists that are accessed at the expensive costs of computing the deviation and accuracy. Under that setting, while the S-list is generated incrementally, two values are maintained (as in [70, 71]): 1) U_{seen}: the maximum utility seen among all binned views generated so far, and 2) U_{max}: a threshold on the maximum possible utility for the binned views yet to be generated. These two values enable efficient navigation of the search space by pruning a significant number of possible binned views as well as utility evaluations, which is achieved by means of two simple techniques:
Incremental Evaluation: The main idea is to calculate the different components of the utility function $U(V_{i,b})$ incrementally and terminate the calculation once it is clear that $V_{i,b}$ is not the optimal binned view. To achieve this, when a candidate binned view $V_{i,b}$ is considered, its $S(V_{i,b})$ value is compared to the maximum utility seen so far, (i.e., U_{seen}), then the calculation of its $D(V_{i,b})$ and $A(V_{i,b})$ values are eliminated (i.e., pruned) if $\alpha_D + \alpha_A + \alpha_S S(V_{i,b}) \leq U_{seen}$. The idea is that since each of $D(V_{i,b})$ and $A(V_{i,b})$ is bounded to 1.0, then a binned view $V_{i,b}$ that satisfies this condition will never have a utility greater than U_{seen}, which makes evaluating its deviation and accuracy unnecessary. Such view will incur no processing costs since $S(V_{i,b})$ is readily available given b, whereas the calculations of $D(V_{i,b})$ and $A(V_{i,b})$ are pruned.

If the above condition is not satisfied, instead of calculating both $D(V_{i,b})$ and $A(V_{i,b})$, further incremental evaluation is performed. Particularly, MuVE decides an order of evaluation of those two objectives. If $D(V_{i,b})$ is evaluated first, then if $\alpha_D D(V_{i,b}) + \alpha_A + \alpha_S S(V_{i,b}) \leq U_{seen}$, then $V_{i,b}$ is safely pruned without the need for evaluating its accuracy. Alternatively, if $A(V_{i,b})$ is evaluated first, then if $\alpha_D + \alpha_A A(V_{i,b}) + \alpha_S S(V_{i,b}) \leq U_{seen}$, then the deviation objective is not calculated and $V_{i,b}$ is pruned. The evaluation order of these two objectives is very important for pruning of low utility views. To decide the evaluation oder of those two objectives, MuVE applies a simple priority function, such that if:

$$\frac{\alpha_A}{C_t(V_{i,b}) + C_a(V_{i,b})} > \frac{\alpha_D}{C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b})} \tag{3.9}$$

then $A(V_{i,b})$ is evaluated first, otherwise $D(V_{i,b})$ is the one to be evaluated first. The idea is to give higher priority to evaluating an objective if it incurs less processing cost and/or contributes more to the utility function that is to be maximized. Recall that $C_t(V_{i,b})$, $C_c(V_{i,b})$, $C_d(V_{i,b})$, $C_a(V_{i,b})$ are the costs of evaluating the target view, comparison view, deviation, and accuracy, respectively. To estimate such costs for a binned view $V_{i,b}$, MuVE simply maintains a moving average of each of those costs over the previous $V_{i,1}, V_{i,2}, ..., V_{i,b-1}$ binned views. Particularly, whenever a short circuit fails and an objective is evaluated, the cost for evaluating the operations involved in that objective is updated

(a) First Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60					$U_{max} = 0.90$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.87	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(b) Second Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60	**0.65**				$U_{max} = 0.87$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.66	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(c) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60	0.65	0.60			$U_{max} = 0.85$
$D(V_{i,b})$	0.80	0.56	Pruned			
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.74$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	$U_{max} = 0.83$
$D(V_{i,b})$	0.80	0.56	Pruned	0.55	0.52	
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	0.63	0.74	

Figure 3.5: Example: The MuVE Scheme

as: $C_x(V_{i,b}) = \beta C_x(V_{i,b-1}) + \frac{1-\beta}{b-2} \sum_{j=1}^{b-2} C_x(V_{i,j})$, where x is any of the four operations listed above, and $\beta = 0.825$ to give more weight to the most recent costs. In our experiments (Section 3.5), we consider other options for setting the priority function and discuss the trade-offs between those options.

Early termination: when a binned view $V_{i,b}$ is considered for evaluation, the threshold U_{max} is updated to $U_{max} = \alpha_D + \alpha_A + \alpha_S S(V_{i,b})$. That is, assuming that $V_{i,b}$ will receive the maximum score of 1.0 under both the deviation and accuracy objectives. In that case, if $U_{seen} \geq U_{max}$, then it is guaranteed that the actual utility of $V_{i,b}$ cannot exceed U_{seen}. Moreover, since all the following views starting at $V_{i,b+1}$ will have lower S values, they are also guaranteed to provide utilities less than U_{seen}. Hence, those views are pruned and early termination is reached.

Example 5. *Consider applying MuVE search on a non-binned view V_i, which is represented by 5 binned views $V_{i,2}$ to $V_{i,6}$ (Figure 3.5). Further, consider that $k = 1$, $\alpha_D = 0.2$, $\alpha_A = 0.6$ and $\alpha_S = 0.2$. For each binned view $V_{i,b}$, its usability $S(V_{i,b})$ is already known and the binned views are ordered accordingly. However, the values of the other two objectives $A(V_{i,b})$ and $D(V_{i,b})$ are calculated as MuVE progresses. $U(V_{i,b})$ is the utility of the binned view $V_{i,b}$ if all three objectives are known, otherwise, it specifies an upper bound on that utility. U_{seen} is the maximum utility seen so far and U_{max} is the threshold on the maximum possible utility. Figure 3.5 shows a snapshot of how MuVE returns the top-1 binned view for V_i. In the first iteration (Figure 3.5a), U_{max} is set as $\alpha_D + \alpha_A + \alpha_S S(V_{i,2}) = 0.90$,* *then both the accuracy and deviation of the first binned view $V_{i,2}$ are computed (shown as a highlighted column). Since $V_{i,b}$ is the only seen view, therefore, $U_{seen} = U(V_{i,2}) = 0.62$. In the second iteration (Figure 3.5b), MuVE considers the next view (i.e., $V_{i,3}$). The new value of U_{max} is 0.87 calculated by $\alpha_D + \alpha_A + \alpha_S S(V_{i,3})$. According to our priority function, $V_{i,3}$ is probed for accuracy and its new upper bound on utility is computed as $\alpha_D + \alpha_A A(V_{i,3}) + \alpha_S S(V_{i,3}) = 0.66$, which is stored in $U(V_{i,3})$. Since that value of 0.66 is greater than the current value of U_{seen}, then $V_{i,3}$ can possibly have a utility greater than U_{seen}, so it is further probed for deviation. In the third iteration (Figure 3.5c), $V_{i,4}$ is probed for accuracy. Its updated upper bound on utility $U(V_4) = 0.61$, which is less than U_{seen}, therefore its deviation computation is pruned. Similar to $V_{i,2}$, the next two views $V_{i,5}$ and $V_{i,6}$ are also fully probed (both accuracy and deviation are evaluated). Finally, MuVE selects $V_{i,6}$ as the top-1 view, which has the highest utility.*

In Example 5, MuVE managed to prune the one deviation calculation for $V_{i,4}$. In general, the amount of pruning achieved by MuVE depends on several factors including the data distribution and the weights of each objective. To attain even higher pruning power, our uMuVE and MuMuVE schemes are proposed next.

3.4.5 The uMuVE Scheme: Upper Bound Based MuVE

Similar to the TA algorithm, MuVE visits the sequence of possible binned views one at a time. For each visited view $V_{i,b}$, the objective values $A(V_{i,b})$ and $D(V_{i,b})$ are either immediately calculated or pruned. Hence, all the necessary processing is completed once a view is visited and no backtracking is needed. Alternatively, in this section, we present uMuVE, which extends the *Upper Algorithm* [70, 73] and allows for interleaved processing of views. Particularly, uMuVE follows a greedy approach, in which it evaluates the most promising views first so that to minimize the total processing time while providing the same results as our original MuVE.

Like MuVE, uMuVE also maintains U_{max}, a threshold on the highest possible utility of a view in the S-list, which has not yet been visited. Additionally, each binned view $V_{i,b}$ has an *upper bound* $U_{upper}(V_{i,b})$ on its utility, which is the maximum possible utility a view $V_{i,b}$ can have. In uMuVE a *priority queue PQ* is maintained based on the upper bound scores of the views. When a view is retrieved from the S-list, its $U_{upper}(V_{i,b})$ equals to $\alpha_D + \alpha_A + \alpha_S S(V_{i,b})$ and it is added to PQ. Once any or both of the objectives (i.e., deviation or accuracy) is calculated, $U_{upper}(V_{i,b})$ of the view is updated accordingly.

For example, in Example 5, the first view retrieved from the S-list and added to PQ is $V_{i,2}$ (Figure 3.6a). According to our earlier explanation, initially $U_{upper}(V_{i,2}) = U_{max} = \alpha_D + \alpha_A + \alpha_S S(V_{i,2}) = 0.90$. In the first iteration, according to our priority function (Eq. 3.9), accuracy of $V_{i,2}$ is computed and its upper bound on utility is updated to 0.66.

In each iteration, uMuVE decides to process one out of two views: 1) view $V_{i,b+1}$, which is the next view in the S-list, or 2) V_h, which is the view with the highest U_{upper} in PQ. To decide between

(a) First Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.90$
$A(V_{i,b})$	0.60					
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.87	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(b) Second Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.87$
$A(V_{i,b})$	0.60	0.65				
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(c) Fifth Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.83$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.83$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$		Pruned			0.52	
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.72	0.74	

Figure 3.6: Example: The uMuVE Scheme

those two views, the upper bound of V_h is compared against U_{max} and the following two cases are considered:

Case 1: $U_{upper}(V_h) \leq U_{max}$: If the upper bound of V_h is lower than the upper bound of the unseen views and there are still views that are not added to PQ yet. Then, this is an indication that the next view in S-list can have a higher upper bound on its utility as compared to all of the views in PQ. Particularly, U_{upper} of the next binned view $V_{i,b+1}$ is computed and U_{max} is updated as $U_{max} = U_{upper}(V_{i,b+1})$.

To further illustrate that case, consider Figure 3.6 again. As $U_{upper}(V_{i,2})$ is less than U_{max}, the decision on pruning or probing of $V_{i,2}$ is deferred for now (shown as dark highlighted column) and it will be kept in memory until uMuVE comes back to it. In the second iteration (Figure 3.6b), U_{max} is updated according to the new view $V_{i,3}$ and $V_{i,3}$ is added to PQ. After probing $V_{i,3}$ on accuracy, its upper bound on utility also becomes less than U_{max} and uMuVE retrieves the next view from the S-list. In the next three iterations, uMuVE will add the next three views $V_{i,4}$, $V_{i,5}$, $V_{i,6}$ in PQ and probe them for accuracy only, as shown in Figure 3.6c.

Case 2: $U_{upper}(V_h) > U_{max}$ or S-list is empty: If $U_{upper}(V_h)$ is greater than U_{max}, then this is an indication that V_h can be the top-1 view. If V_h is already fully probed then it is returned as the top-1 view and the rest of the views are removed from PQ. However, if it is partially probed then its other objective is calculated and $U_{upper}(V_h)$ is updated accordingly.

If S-list is empty, uMuVE selects the view with highest upper bound V_h, probe it further and update its $U_{upper}(V_h)$. To illustrate this case consider Figure 3.6d, in which uMuVE has added all possible views to the PQ and S-list is empty. Now, uMuVE will select a view from the front of PQ i.e., $V_{i,6}$ and compute its deviation objective. The final utility of $V_{i,6}$ is 0.74 and the upper bound of every other view is less than this utility, therefore, $V_{i,6}$ is returned as top-1 view. Figure 3.6 clearly shows that uMuVE is able to prune three deviation evaluations as compared to MuVE.

3.4.6 The MuMuVE Scheme: Memory-aware uMuVE

Our machines have limited computational and memory resources. Therefore, along with the processing cost of proposed search schemes, it is also important to understand their memory requirements. For instance, consider our proposed schemes MuVE and uMuVE, MuVE's memory usage is negligible since it considers only one view at a time. However, uMuVE stores multiple views in the priority queue, therefore, it uses more memory and its memory requirements need to be quantified and optimized. In the classical Upper scheme [70, 73], objects are points in multidimensional space and partial probes only require to store single value per dimension. Therefore, memory is not a critical problem in that scenario. However, in our case, each object is a view and view data is stored in memory. This brings forth the addition challenge of optimizing memory requirements of uMuVE. In this section, we address that challenge by quantifying the memory needs of uMuVE and proposing the memory-aware uMuVE scheme (MuMuVE).

As explained in Section 3.4.5, the decision on the views in the priority queue remain pending, unless there is an indication that those views should be probed further or pruned. The memory requirements for uMuVE depend on the order in which views are visited. For instance, for a non-binned view V_i, if the top-1 binned view is seen earlier in the search, memory needs will be minimal. In the worst case scenario, the data for every binned view $V_{i,b}$ is stored in memory. The amount of memory $M_U(V_{i,b})$ used by a binned view $V_{i,b}$ equals to the number of bins b of $V_{i,b}$. Then, a binned view $V_{i,b}$ in PQ, can be in one of the following three states:

1. $A(V_{i,b})$ & $D(V_{i,b})$ are known: That is, view $V_{i,b}$ has been fully probed and both objectives have been evaluated. Therefore, the data for comparison and target view is not required anymore. Hence, $M_U(V_{i,b}) = 0$.

2. $A(V_{i,b})$ is known & $D(V_{i,b})$ is unknown: That is, the accuracy of view $V_{i,b}$ has been computed. Specifically, the underlying target view of $V_{i,b}$ has been retrieved and it will be kept in memory until $V_{i,b}$ is pruned or probed for deviation. Therefore, $M_U(V_{i,b}) = b$.

3. $D(V_{i,b})$ is known & $A(V_{i,b})$ is unknown: That is, the deviation of view $V_{i,b}$ has been evaluated. Particularly, the underlying target and comparison views of $V_{i,b}$ were retrieved. After computing deviation, the comparison view is discarded, but the target view will remain in main memory because it may be needed if uMuVE decides to evaluate the accuracy of this view. Therefore, $M_U(V_{i,b}) = b$.

(a) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 9$
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(b) Fourth Iteration: Initial State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	**0.60**			$M_U(V_i) = 5$
$D(V_{i,b})$			**0.54**			
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	**0.52**	$\leq$0.84	$\leq$0.83	

(c) Fourth Iteration: Final State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80		$M_U(V_i) = 10$
$D(V_{i,b})$			0.54			
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	0.52	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$		Pruned	0.54	0.55	0.52	
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	0.52	0.63	0.74	

Figure 3.7: Example: The MuMuVE Scheme (Max-Bins)

Let L_i be the range of the dimension attribute of the non-binned view V_i, then the maximum memory needed to search for the top-1 binned view is:

$$M_U(V_i) = \sum_{b=2}^{L_i} M_U(V_{i,b}) = \sum_{b=2}^{L_i} b$$

In Example 5, the amount of memory used by uMuVE (Figure 3.6) is $M_U(V_i) = \sum_{b=2}^{6} b = 2+3+4+5+6 = 20$. This is because uMuVE had all 5 views in the memory.

Clearly, uMuVE always has lower processing cost as compared to MuVE, but it has additional memory needs. To balance the trade-off between memory and processing time, a memory-aware version of uMuVE (MuMuVE) is proposed next.

MuMuVE is an extension of uMuVE for memory-bounded evaluation of views. The new scheme attempts to minimize view probes for a given amount of memory M_L. The main idea is when the memory utilization gets close to M_L, instead of adding more views to PQ, give preference to probing those views which are already in PQ. Consequently, memory will be evicted.

As explained previously, uMuVE selects a view V_h with the highest U_{upper} from PQ and if $U_{upper}(V_h) < U_{max}$, moves a binned view V_S from S-list to PQ. However, for MuMuVE before adding V_S to PQ, we also need to check if there is enough space available. Therefore, the condition is modified as: if $U_{upper}(V_h) < U_{max}$ & $M_U(V_S) < M_L$ then add V_S to PQ. V_S is not added to PQ if any of the two conditions is violated. Hence, if $U_{upper}(V_h) < U_{max}$ is false, MuMuVE probes V_h further. If $M_U(V_S) < M_L$ is false, that means $M_U(V_i)$ has reached the bound M_L. MuMuVE will choose a partially probed view $V_{h'}$ from PQ to complete its utility evaluation. This ensure that memory occupied by $V_{h'}$ is evicted and more views can be added to PQ. The decision about which view to evict is critical, as it affects the processing time. We consider the following three options:

1) *Max-Bins*: In this option, MuMuVE chooses the view with the maximum number of bins from the partially probed views in PQ. Although, there is no indication that this view might be the top-1 view, however, it ensures that the maximum possible memory is evicted by a single probe.

Again, consider Example 5 and assume $M_L = 10$. MuMuVE starts adding views from S-list to PQ as in uMuVE. By the end of the third iteration (Figure 3.7a), $V_{i,2}$, $V_{i,3}$ and $V_{i,4}$ are already in PQ. The amount of memory used $M_U(V_i)$ is 9. The next view in S-list $V_{i,5}$ require 5 locations while there is only 1 available, therefore, MuMuVE needs to evict memory before adding $V_{i,5}$ to PQ. According to max-bins, MuMuVE probes the view having maximum bins i.e., $V_{i,4}$ (Figure 3.7b). Consequently, there is enough space to add $V_{i,5}$ as shown in the final state of the fourth iteration in Figure 3.7c. After the final iteration (Figure 3.7d), this scheme is able to prune two objective evaluations as compared to four of uMuVE. However, in worst case it only used 50% of the space compared to uMuVE.

2) *Max-Utility*: In this option, MuMuVE chooses the view $V_{h'}$ having maximum upper bound U_{upper}. However, if number of bins of $V_{h'}$ is lower than the number of bins of V_S then it will keep choosing $V_{h'}$ views until there is enough space. Therefore, as result MuMuVE evaluates unnecessary objectives and that will increase processing cost as compared to MuVE. On the other side there is possibility that the view MuMuVE has fully probed might be the top-1, which will end the search.

Consider Example 5 again and assume $M_L = 10$. Similar to max-bins, $V_{i,2}$, $V_{i,3}$, $V_{i,4}$ are added to PQ and M_L is reached (Figure 3.8a). According to max-utility, the view having maximum upper bound on utility i.e., $V_{i,2}$ is probed further (Figure 3.8b). However, the available space $(M_L - M_U)$ is still not enough to accommodate $V_{i,5}$. Therefore, MuMuVE probe $V_{i,3}$ (Figure 3.8c), which will reduce the used memory space M_U to 4. Then, $V_{i,5}$ can be added to PQ as shown in the final state of fourth iteration in Figure 3.8c. Max-utility pruned one objective evaluation as compared to two of max-bins.

3) *Max-Min-Utility:* Memory requirements and processing time of the max-utility scheme can be further reduced by keeping track of lower bound on the utility of the views in PQ. Particularly, when the memory is full, view $V_{h'}$ with the maximum utility from PQ is probed further. Utility of this $V_{h'}$ becomes lower bound on the utility i.e., $U_{lower} = U(V_{h'})$ and all views $V_{i,b}$ from PQ which have $U_{upper}(V_{i,b}) < U_{lower}$ are removed, because none of these can be top-1 view.

Consider Figure 3.8 again, and assume U_{lower} is also maintained. In initial state of the fourth iteration (Figure 3.8b) after view $V_{i,2}$ is fully probed the lower bound on utility is updated as $U_{lower} = 0.62$. As the upper bound on the utility of $V_{i,4}$ is less than U_{lower}, which means $V_{i,4}$ can not be in top-1 therefore it is removed from PQ. Now, there is enough memory available to probe the next view. After the final step, max-min-utility is able to prune two objective evaluations compared to max-utility.

(a) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 9$
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(b) Fourth Iteration:Initial State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 7$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(c) Fourth Iteration: Final State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80		$M_U(V_i) = 4$
$D(V_{i,b})$	0.80	0.56				
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$	0.80	0.56	Pruned	0.55	0.52	
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	0.63	0.74	

Figure 3.8: Example: The MuMuVE Scheme (Max-Utility)

3.4.7 Vertical Search Schemes

Recall that the goal of this work is to recommend the top-k visualizations that maximize our multi-objective utility function. In the previous section, we discussed horizontal search strategies, which find the optimal binned $V_{i,opt}$ for a given non-binned view V_i. As discussed earlier, the space of possible non-binned views, is of size $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. In the case where $\mathbb{A}$ is a set of numerical dimensions, then the total number of corresponding possible binned views is N_B, where $N_B = \sum_{j=1}^{|\mathbb{A}|} 2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$. Hence, the goal is simply to find the top-k binned views across those N_B views. We note, however, that recommending two different binned views that correspond to the same non-binned views adds little value to the analyst and is rather redundant. Hence, if V_{x,b_1} and V_{y,b_2} are two views in the top-k list, then $x \neq y$. Consequently, we propose the following vertical search strategies.

In our first strategy for vertical search, we extend linear search (as described in the previous section) for the purpose of finding the top-k recommendations. Particularly, in this simple strategy, the set of all possible non-binned views $\mathbb{V}$ is traversed sequentially in an exhaustive manner. Then, each view $V_i \in \mathbb{V}$ is expanded and searched horizontally to find its optimal binned view $V_{i,opt}$. As linear search finishes scanning $\mathbb{V}$, the optimal binned view corresponding to each view V_i is identified, and out of those, the k with the highest utility are the ones to be recommended.

Note, however, that under this vertical linear search, the vertical and horizontal searches are clearly decoupled. Hence, while the vertical search is performed linearly, the choice of the horizontal search strategy is open. Given the algorithms discussed so far, this allows for the combinations denoted as follows: *linear-linear:* in which linear search is used for both the vertical and horizontal searches, and *MuVE-Linear:* in which linear search is used for vertical search, whereas the optimized MuVE, as described in the previous section, is used for horizontal search. Obviously, the latter combination allows leveraging the optimizations offered by MuVE to reduce the cost of each horizontal search. Towards further optimizations, in the following we discuss extending MuVE, uMuVE and MuMuVE to perform both the vertical and horizontal searches (i.e., MuVE-MuVE, uMuVE-Linear, uMuVE-uMuVE, MuMuVE-Linear, MuMuVE-MuMuVE).

Extending MuVE to perform both horizontal and vertical searches is straightforward. To explain that extension, recall that for performing horizontal search on a non-binned view V_i over a numerical dimension of range L, MuVE progressively populates the S-list with the values $\alpha_S S(V_{i,1}), \alpha_S S(V_{i,2}), ..., \alpha_S S(V_{i,L})$, which are the values of the usability objective sorted in decreasing order. Hence, to allow vertical search, MuVE traverses the set of non-binned views $\mathbb{V}$ in a round-robin fashion, where in a round r each view $V_i \in \mathbb{V}$ is appended to the S-list as $V_{i,r}$, given that r is less than the maximum number of bins that is possible for that view. Adding a view $V_{i,r}$ to the S-list triggers evaluating the multi-objective utility function $U(V_{i,r})$. That evaluation is performed similar to the one described above for the horizontal search, except that the pruning conditions employed for the incremental evaluation are set for top-k instead of top-1. Evaluating new views continues until all possible binned views are generated or until early termination is reached, then the top-k views with the highest utility are returned to the user.

In comparison to MuVE-Linear described above, using MuVE for both vertical and horizontal search clearly offers further reductions in cost by means of increasing the number of pruned operations. To explain this, consider an uninteresting view V_i (i.e., a view with low deviation). If that view is considered in isolation, as in MuVE-Linear, then significant processing time will be spent on finding $V_{i,opt}$. However, $U(V_{i,opt})$ is expected to still be very small compared to the other views, which are more interesting. Under MuVE, however, those interesting views will lead to increasing the value of U_{seen}, which in turn allows for pruning many of the objective evaluations that were to be performed on V_i.

Extending uMuVE to perform vertical and horizontal search is similar to MuVE. Particularly, like MuVE, uMuVE traverses the set of non-binned views $\mathbb{V}$ in round robin fashion and in each round r it appends each $V_{i,r}$ to S-list. However, uMuVE also maintains a priority queue PQ and it moves views from S-list to PQ when none of the views in PQ are promising enough to be probed further. When the utility of a fully probed view V_h becomes greater than U_{max} it is recommended as top-k and removed from PQ. Consequently, all of the views with same A, M and F as V_h are also removed from PQ to ensure that the recommended views are diverse.

Progressive Results: A key advantage of using uMuVE in both horizontal and vertical direction is that it can produce *progressive results*, i.e., results as they become available rather than waiting for complete evaluation of all of the views. Specifically, after returning the top-1 view, if more views are required uMuVE continues to probe more views and complete their utility evaluations. Then, uMuVE

returns the next view with the highest utility and it keeps going until all of the top-k views are returned. This feature can be particularly beneficial in an interactive data exploration scenario, where as soon as the first few results are known, the user may decide to terminate the current search and begin a next search by changing some of the input parameters.
Additional Aggregate Queries: We note that while in this thesis we consider only aggregation with a single numeric dimension (i.e., single group-by attribute) , our techniques are directly applicable to the more general scenario, in which there are multiple numeric group-by attributes. Such aggregations will result in multi-column views that can be visualized as multi-dimensional or stacked bar charts. Hence, more memory is required to store those multi-column views but the process of the objective computation would remain the same.

Another interesting setting to consider is when some of the possible aggregations have a numerical dimension attribute, whereas others are based on a categorical dimension attribute. In that case, recommending the top-k interesting views is rather challenging as it requires the employed utility function to be able to fairly compare the utility provided by those two different kinds of views. One idea is to adapt our multi-objective function (Eq. 3.5) such that if a view V_i is based on a categorical dimension, then its overall utility is computed as follows: 1) the deviation of V_i is computed as in Eq. 3.2, 2) the accuracy of V_i is always equal to 1.0 since no summarisation is performed, and 3) the usability of V_i is also equal to 1.0 since categorical values cannot be aggregated in larger bins. However, that simple adaptation is expected to always assign high utility values to those views based on categorical dimensions because their accuracy and usability will always receive a perfect score of 1.0. This is clearly in contrast with our goal of ensuring a fair comparison between different kinds of views. Hence, our utility function needs to incorporate additional measures to enforce that fairness (e.g., diversification [74]). A detailed exploration of that problem is part of our future work.

3.4.8 Approximate Search Schemes

All the search algorithms presented so far, except for Hill Climbing, are accurate in the sense that they provide the same top-k views as the baseline exhaustive linear search. Meanwhile, Hill Climbing, being a local search algorithm, is prone to hit some local maxima when used for horizontal search, hence recommending views with lower utility. The degree of inaccuracy exhibited by local search methods is typically unpredictable and highly depends on the behavior of the utility function to be optimized. To the contrary, MuVE employs optimization techniques that allow for significant performance gains, while at the same time providing the same recommendations as the exhaustive baseline. In the following, we introduce several approximations to the MuVE scheme to further improve its performance, while incurring negligible loss in the quality of recommendation.

View Refinement

In this approximation, instead of horizontally expanding and searching each and every non-binned view V_i to find its optimal binned view $V_{i,opt}$, only a small number of views are selected for that expensive horizontal search. The main idea is to perform a simple vertical search to quickly select that small set of views, which are then refined using horizontal search to find the top-k recommended views. To achieve this, the set of non-binned views $\mathbb{V}$ is searched vertically using a predefined number of bins, which is the same for all views. Specifically, the utility of each non-binned view $V_i \in \mathbb{V}$ is computed as $U(V_{i,def})$, where def is the same value for all views. Then the top-k views according to that binning def are selected, which is easily performed using linear search or MuVE. Consequently, utility of each one of those k selected views (i.e., $U(V_{i,def})$) is further refined using horizontal search to find its optimal binning (i.e., $V_{i,opt}$). Hence, only k views are selected for horizontal search, which can be applied using linear search, MuVE, or HC, as explained earlier. We note that the default binning value def is a system parameter. Our experimental evaluation shows that choosing a moderate number of bins results in significant reductions in cost, while at the same time providing high fidelity recommendations.

View Skipping

The main idea underlying that approximation is to skip the horizontal search for some non-binned views, thus saving the costs incurred in finding their $V_{i,opt}$. To better understand this idea, recall that each non-binned view V_i is basically represented by a tuple (A,M,F) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, and $F \in \mathbb{F}$. Hence, for each numerical dimension A, there exists a set of views $\mathbb{V}_A$, which share the same dimension A, while being defined using different measures and aggregate functions, such that $|\mathbb{V}_A| = |\mathbb{M}| \times |\mathbb{F}|$. Accordingly, in this approximation we assign the same binning to all the views in the set $\mathbb{V}_A$. To find that binning (i.e., number of bins), a view V_i is selected arbitrary from the set $\mathbb{V}_A$. For that non-binned view, horizontal search is performed normally using any of the strategies outlined above (i.e., linear, HC, or MuVE), so that to find its corresponding $V_{i,opt}$. Then that optimal number of bins opt is assigned to all views in $\mathbb{V}_A$, and their utilities are evaluated accordingly. The premise is that the range of a numerical dimension A is an important factor in deciding its optimal binning. Hence, since all views in $\mathbb{V}_A$ share the same dimension A, then the optimal binning of those views will have a very small variance from some mean value, which is selected as described above and used to represent the set of views $\mathbb{V}_A$. Accordingly, the number of times horizontal search is invoked is equal to the number of numerical dimensions (i.e., $|\mathbb{A}|$). Those horizontal searches are easily integrated with one of the vertical searches described above (i.e., linear or MuVE).

Range Partitioning

The idea for this approximation is to reduce the complexity of the horizontal search based on simple range partitioning techniques. Recall that a non-binned view $V_i = (A,M,F)$ is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,b}, ..., V_{i,L}\}$, where b is the number of bins, and L is the range of the continuous numerical dimension A. By default, MuVE as well as linear search, assume the value of b to be continuous in the range $[1-L]$ with incremental additive step of 1.0, leading to corresponding binning of widths: $\frac{L}{1}, \frac{L}{2}, ..., 1$. Hence, a numerical dimension A with a large continuous range results in a

Datasets	Number of tuples	Range of A_1	Range of A_2	Range of A_3	Number of Views
DIAB	768	21-81	0-67	0-199	2,961
NBA	651	19-38	1-83	1-2981	27,765
CENSUS	32,561	17-90	1-16	1-99	1,701

Table 3.2: Details of Datasets

large number of binning options, and in turn a high cost for performing horizontal search. Accordingly, we employ two simple alternative methods for range partitioning, namely: 1) additive, and 2) geometric. The default partitioning described above is an instant of the additive method, in which the incremental step s is set to 1. In the general case, the incremental step s is a parameter, hence, a non-binned view V_i is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,1+s}, V_{i,1+2s}, ..., V_{i,L}\}$. Alternatively, in the geometric method (e.g., [75]), V_i is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,2^0}, V_{i,2^1}, V_{i,2^2}, ..., V_{i,L}\}$. Naturally, both methods are expected to reduce the processing time incurred in horizontal search, at the expense of some reduction in the fidelity of recommendation, which will be evaluated experimentally in the next sections.

3.4.9 Experimental Testbed

We perform extensive experimental evaluation to measure the efficiency of the different top-k view recommendation strategies presented in this thesis. Here, we present the different parameters and settings used in our experimental evaluation.

Setup: We built a platform for recommending visualizations, which extends the SeeDB codebase [6] to support numerical dimensional values, binned aggregation, and the different search strategies presented in this thesis. Our experiments are performed on a Corei7 machine with 16GB of RAM memory. The platform is implemented in Java, and PostgreSQL is used as the backend database management system.

Schemes: We investigate the performance of the different combinations of the vertical and horizontal search strategies presented in this thesis. Our naming convention for those combinations is represented as: *SearchH-SearchV*, where *SearchH* denotes the search strategy employed for horizontal search, whereas *SearchV* is the one for vertical search. This leads to the following combinations: *Linear-Linear*, *MuVE-Linear*, *MuVE-MuVE*, *uMuVE-Linear*, *uMuVE-uMuVE*, *MuMuVE-Linear* and *MuMuVE-MuMuVE*. For instance, in *MuVE-Linear*, MuVE is used for horizontal search, whereas linear search is applied for vertical search. In the presence of approximation, as discussed in Section 3.4.8, we extend our notation to: *SearchH(AppH)-SearchV(AppV)*. Hence, the possible horizontal approximations are: *SearchH(A)*, and *SearchH(G)*, which denote the additive and geometric range partitioning, respectively. For vertical approximations, *SearchV(R)* denotes the view refinement approximation, and *SearchV(S)* is for the view skipping approximation.

Data Analsyis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. We use three datasets: *DIAB:* dataset of diabetic patients [76], *NBA:* dataset of basketball players [77] and *CENSUS:* dataset of adult census income [78]. The independent numeric attributes of each dataset are used as dimensions, whereas the observation attributes are used as measures. For instance, in the DIAB dataset, dimensions are selected from age, BMI, etc., whereas measures are selected from insulin level, glucose concentration, etc.

The DIAB, NBA and CENSUS datasets have 9, 28 and 15 attributes, respectively. In our default setting, we select 3 dimensions, 3 measures, and 3 aggregate functions. Table 3.2 shows the range of each dimension A for every dataset and accordingly the number of possible views are also shown. In the analysis, all the α values are in the range $[0-1]$, where $\alpha_D + \alpha_A + \alpha_S = 1$. In the default setting, $\alpha_D = 0.2$, $\alpha_A = 0.2$, $\alpha_S = 0.6$ and $k = 5$, unless specified otherwise. The input queries for each dataset are: DIAB: `SELECT * FROM DIAB WHERE Pregnancies>3`, NBA: `SELECT * FROM NBA WHERE team=GSW`, and CENSUS: `SELECT * FROM CENSUS WHERE income>50K`.

Performance: We evaluate the efficiency of the different recommendations strategies in terms of (1) *Cost:* As mentioned in Section 4.4, the cost of a strategy is the total cost incurred in processing all the candidate binned views, which is measured in wall clock time, and (2) *Fully Probed Views:* Count of the views $V_{i,b}$ for which both objectives $D(V_{i,b})$ and $A(V_{i,b})$ were calculated. Each performance metric is reported based on the average of 10 different executions. (3) *Fidelity:* It is a measure of the degree of accuracy achieved by a certain scheme. Particularly, if $\mathbb{V}_{opt}$ is the set of top-k views recommended by a baseline optimal scheme, whereas $\mathbb{V}_{rec}$ is the set of views recommended by an approximated scheme, then fidelity is measured to capture the difference between the sum of the utilities offered by $\mathbb{V}_{opt}$ and $\mathbb{V}_{rec}$. Formally:

$$F = 1 - \frac{U(\mathbb{V}_{opt}) - U(\mathbb{V}_{rec})}{U(\mathbb{V}_{opt})} \tag{3.10}$$

3.4.10 Experimental Evaluation

In the following experiments, we evaluate the performance of both our optimization techniques (Section 3.4.10), as well as our approximation techniques (Section 3.4.10), under different parameter settings.

Optimization Techniques

Impact of the α parameters (Figures 3.9, 3.10 and 3.11): In this set of experiments, we measure the impact of the α values on processing time (i.e., cost). Figures 3.9, 3.10 and 3.11 show how the cost of the different schemes is affected by changing the values of α_D, α_A and α_S.

In Figures 3.9 and 3.10, α_S is set to constant 0.2 while α_A and α_D are changing. In particular, as shown in the figures, α_D is increased, while α_A is implicitly decreased and is easily computed as $\alpha_A = 1 - \alpha_D + \alpha_S$. Figure 3.9 shows that *Linear-Linear* has almost same cost for all values of α_S, which is expected since it performs exhaustive search over all combinations of A, M, F, and B. Therefore, its cost depends on the number of all possible combinations, irrespective of the values of α.

Figure 3.9 also shows that *MuVE-MuVE* has lower cost than *Linear-Linear* and *MuVE-Linear*, especially in the region where α_D is low and correspondingly, α_A is high. This is because interesting

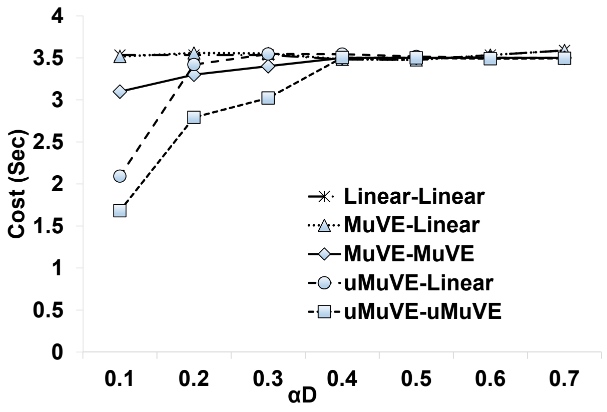


Figure 3.9: DIAB:Impact of α_A and α_D on cost, while $\alpha_S = 0.2$

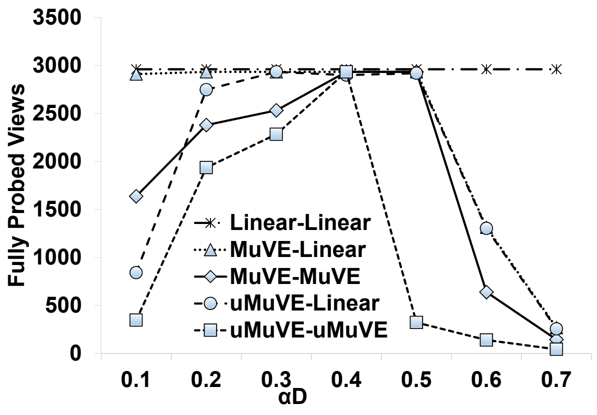


Figure 3.10: Impact of α_A and α_D on fully probed views, while $\alpha_S = 0.2$

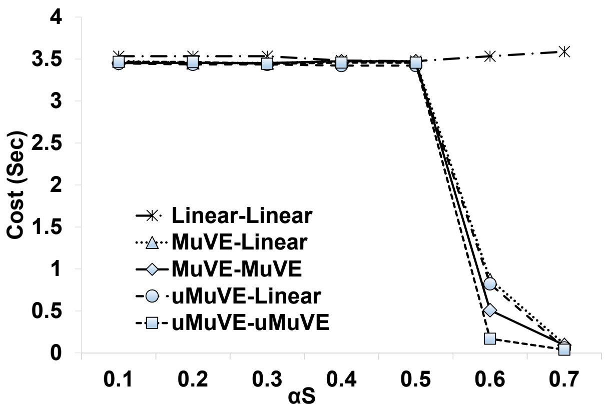


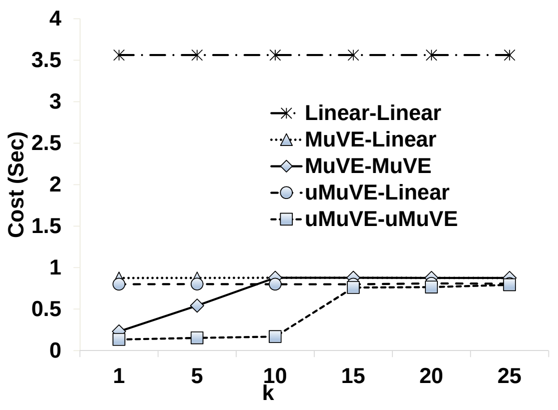
Figure 3.11: DIAB: Impact of α_D and α_S on cost, while $\alpha_A = 0.2$

views with high accuracy will lead to a higher U_{seen}, which in turn allows for pruning the less interesting views during the vertical search. Furthermore, Figure 3.9 also shows that *uMuVE-uMuVE* offers the lowest cost, especially when $\alpha_D < 0.4$. For instance, at $\alpha_D = 0.2$, *uMuVE-uMuVE* has almost 30% lower cost as compared to *MuVE-MuVE*. This is because after evaluating one objective of a view, *uMuVE* delays probing that view in full until the partial utility of that view becomes higher than the current U_{max}. Hence, it prunes many unnecessary deviation evaluations. The reduction in cost of the *MuVE* variants can be further understood using Figure 3.10, in which we plot the number of views that are probed in full (i.e., both deviation and accuracy are evaluated). Figure 3.10 shows that *MuVE-MuVE* and *uMuVE-uMuVE* fully probe a very low number of views at the high values of α_D. Interestingly, however, that large reduction in the number of probed views does not translate into cost saving as it has been the case at high α_A (Figure 3.9). This is because at high α_D, *MuVE-MuVE* and *uMuVE-uMuVE* mainly prune the operations for computing accuracy, whereas at high α_A mainly the operations for computing deviation are pruned, which typically incur higher processing cost than that needed for computing accuracy.

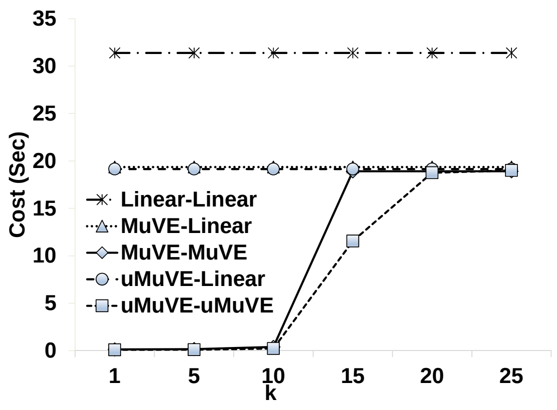
In Figure 3.11, $\alpha_A = 0.2$, whereas α_S is increasing and accordingly α_D is decreasing. Figure 3.11 shows the effect of changing α_S and α_D values on cost. Particularly, Figures 3.11 shows that the MuVE schemes have almost same cost as *Linear-Linear* for smaller values of α_S, but outperform it as the value of α_S increases. For instance, in Figure 3.11 at $\alpha_S > 0.5$, all four schemes show more than 70% reduction in cost as compared to *Linear-Linear*. This happens because in the *MuVE* and *uMuVE* schemes, when α_S is high, there are more chances of applying the short circuiting and early termination conditions based on the usability value, and in turn pruning many of the operations required for evaluating deviation and accuracy. The amount of achieved pruning is further increased under *MuVE-MuVE* and *uMuVE-uMuVE*, which is able to prune those operations during both the vertical and horizontal searches. For instance, Figure 3.11 shows that *uMuVE-uMuVE* reduces the processing cost by almost 75%, compared to *uMuVE-Linear*, at $\alpha_S = 0.6$.

Impact of k (Figure 3.12): In the previous experiments, the value of k is set to 5 (i.e., top-5 views are recommended). Figure 3.12 shows that *Linear-Linear*, *MuVE-Linear* and *uMuVE-Linear* are all insensitive to the increase in the value of k. This is because *Linear-Linear* is exhaustive search, whereas *MuVE-Linear* and *uMuVE-Linear* also performs an exhaustive vertical search. For instance, in Figure 3.12a, in case of top-1 *MuVE-MuVE* reduces the cost by up to 90% compared to the *Linear-Linear* scheme, while the reduction offered by *uMuVE-uMuVE* is up to 85% compared to the *MuVE-MuVE* scheme.

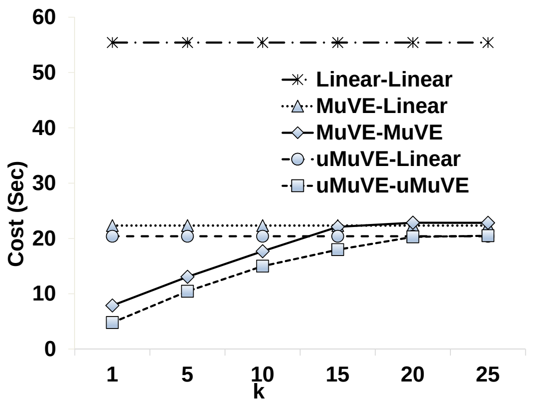
Priority Function Analysis (Figure 3.13): As mentioned in Section 3.4.4, we consider different options for setting our priority function for ordering the evaluation of objectives. The options that we considered are: 1) *Random:* Randomly chooses the objective to evaluate first, 2) *Deviation-First:* Always computes the deviation objective first, 3) *Accuracy-First:* Always computes the accuracy objective first, 4) *Weights-Based:* Selects the objective which has more weight as it contributes more to the objective function, and 5) *Hybrid:* Selects an objective based on its weight and evaluation cost (as in Eq. 3.9).



(a) DIAB



(b) NBA



(c) CENSUS

Figure 3.12: Impact of k on Cost

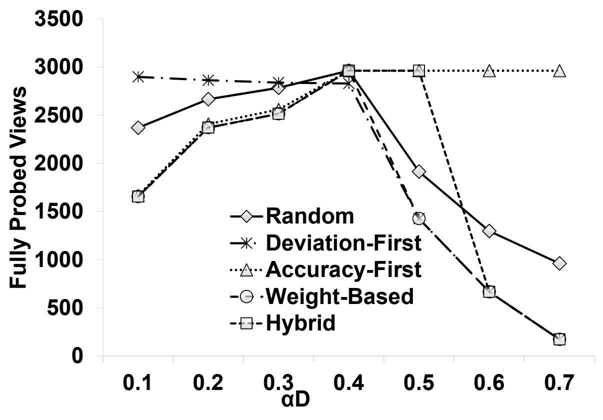


Figure 3.13: Priority Function Analysis

Figure 3.13 shows the cost of *MuVE-MuVE* in terms of the number of fully probed views when incorporating each of the options listed above. In Figure 3.13, $\alpha_S = 0.2$, whereas α_D is increasing and accordingly α_A is decreasing.

Figure 3.13 shows that for low value of α_D, number of fully probed views for accuracy-first scheme are lower than deviation-first because of the short circuiting of deviation objective evaluation. However, for high values of α_D, number of fully probed views for deviation-first are lower because of the short circuiting of accuracy objective evaluation. Hybrid scheme captures the advantage of

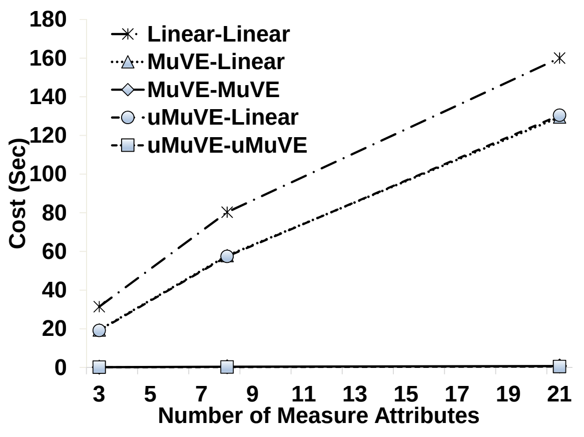


Figure 3.14: NBA: Scalability

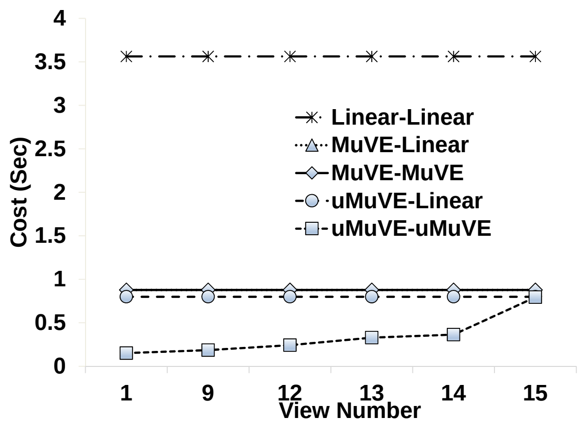


Figure 3.15: DIAB: Progressive Results

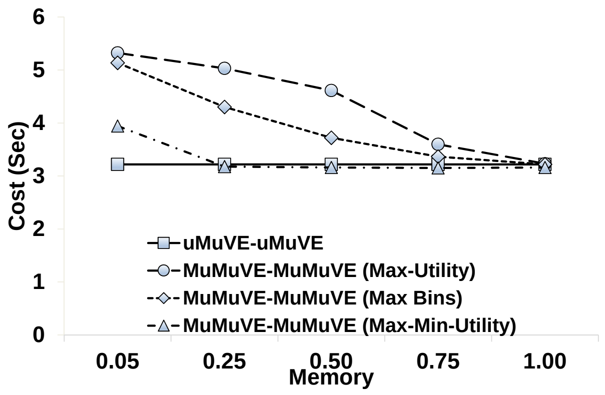


Figure 3.16: Impact of M_L on MuMuVE Scheme

both deviation-first and accuracy-first schemes. For values where α_A is high, the number of database probes are smaller as compared to low α_A. This happened due to the satisfaction of short circuiting and early terminating conditions and as a result pruning of deviation evaluation operations. The deviation-first has the similar trend for higher values of α_D. Here, accuracy evaluation operations are pruned. The figure shows, the random scheme probes mores views as compared to accuracy-first and deviation-first. Therefore, it is not the optimal priority function. Weight-based captures the advantage of both deviation-first and accuracy-first schemes. Hybrid integrates the cost of computing objectives into weights-based priority function. As deviation costs almost twice as accuracy, accuracy gets a preference to get computed first for $\alpha < 0.6$.

Scalability (Figure 3.14): From Section 3.4.1, the theoretical complexity of our recommendation problem is linear in terms of the number of dimensions A, expressed as cA, where c is the product of number of measures, aggregate functions and bin settings. While such complexity applies to both *Linear*, *MuVE* and *uMuVE*, in practice, however, c is much smaller for *MuVE* and *uMuVE* due to pruning. For example, Figure 3.14 shows our results on the NBA data, it can be inferred that c for *Linear* goes up to $\sim$12, whereas it is only $\sim$0.05 for *MuVE* and *uMuVE*.

Progressive Results (Figure 3.15): In this experiment we demonstrate uMuVE's ability to produce results in a progressive fashion. Particularly, Figure 3.15 shows the delay (i.e., processing cost) until producing the i^{th}-top view when recommending a total of 15 views (i.e., top-k, where $k = 15$). As the figure shows, *Linear-Linear*, *MuVE-Linear*, *MuVE-MuVE* and *uMuVE-Linear* produce the top-15 views all together as a batch. However, *uMuVE-uMuVE* produces the 1^{st} top view as soon as it is identified, and then it keeps producing more views in descending order of their utility values. For instance, *uMuVE-uMuVE* recommends the 9^{th} view after only 0.18sec, while *MuVE-MuVE* recommends it along with all the 15 top-k views after 1sec.

Memory Requirements (Figure 3.16): In this experiment we study the performance of our memory-aware MuMuVE scheme under a predefined memory constraint M_L. We particularly evaluate the different variants of MuMuVE, namely: Max-bins, Max-utility, and Max-Min-Utility, against uMuVE. To do that, we set the limit M_L as a percentage of the memory used by uMuVE in the worst case (as shown on the x-axis in Figure 3.16). For instance, a value of 0.75 means that the memory constraint M_L is set to 75% of the maximum memory used by uMuVE. As the figure shows, *uMuVE-uMuVE* acts as a baseline and its performance is independent of M_L. Further, the figure also shows that as M_L is decreased, the cost of all the memory-aware schemes increase. This is because the memory-aware schemes are forced to evaluate extra objectives to reclaim space. The figure shows that Max-Bins outperforms Max-Utility because of its ability to avoid unnecessary probes and reclaim space, whereas Max-Min-Utility performs as good as *uMuVE-uMuVE* while keeping the memory usage well under constraint.

Approximation Techniques

Impact of Additive Range Partitioning (Figures 3.17 and 3.18): In Figures 3.17 and 3.18 we show the impact of having different values of $step$. As expected, Figure 3.17 shows that the *HC-Linear* search scheme provides the same cost regardless of the employed step. This is simply because *HC* employes its own stepping method, as explained earlier. Meanwhile, the cost of *Linear(A)-Linear* decreases with the increase in $step$. This is because when $step > 1$, the search space is reduced by a factor of $step$, which results in that reduction of cost. The figure also shows that *MuVE(A)-Linear* has low cost at $step = 1$ and after that its cost is almost the same as *Linear(A)-Linear*. This is because, when $step = 1$ *MuVE(A)-Linear* gets the opportunity of short circuits and early terminations, which are activated because of the high utility provided by those views with relatively small number of

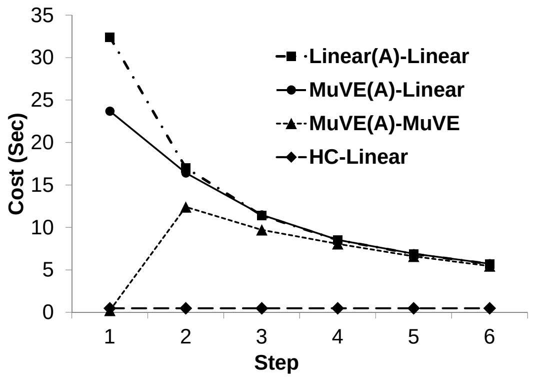


Figure 3.17: NBA: Impact of additive range partitioning on cost

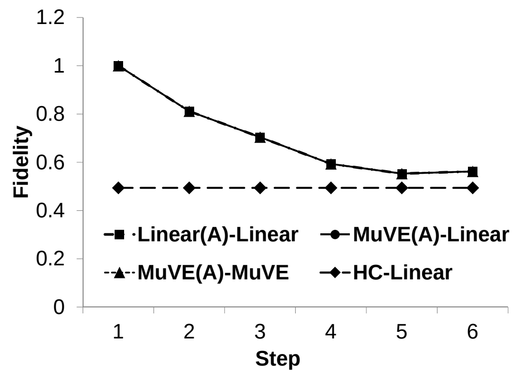


Figure 3.18: NBA: Impact of additive range partitioning on fidelity

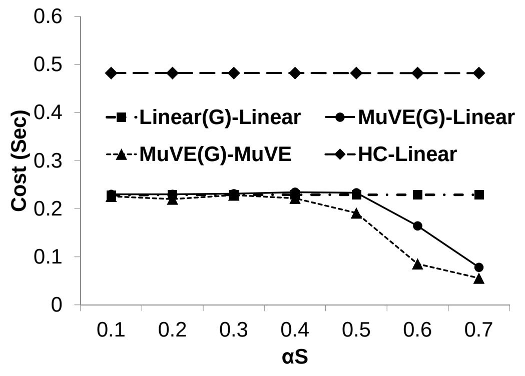


Figure 3.19: NBA: Impact of geometric range partitioning on cost

bins. With higher values of *step*, this opportunity is missed and *MuVE(A)-Linear* behaves almost the same as *Linear(A)-Linear*. Figure 3.18 shows the impact of *step* on fidelity. For *HC-Linear*, similar to its cost, its fidelity is insensitive to *step* and it always provides less than 50% fidelity. This is because *HC-Linear* is a local search based scheme, which is expected to hit a local maxima, hence, failing to achieve the global maxima, which results in low fidelity. Meanwhile, *Linear(A)-Linear*, *MuVE(A)-Linear* and *MuVE(A)-MuVE* show the same pattern with increasing the *step* value.

Impact of Geometric Partitioning (Figure 3.19 and 3.20): For this set of experiments we set $\alpha_A = 0.2$ and observe the effect of changing α_S on the fidelity and cost of when employing geometric partitioning. Figure 3.19 shows that the cost of *Linear(G)-Linear* remains constant because it exhaustively searches for maximum utility view. Meanwhile, geometric partitioning reduces the cost of *MuVE(G)-Linear*

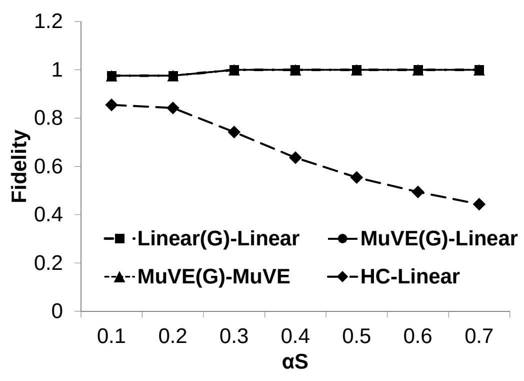


Figure 3.20: NBA: Impact of geometric range partitioning on fidelity

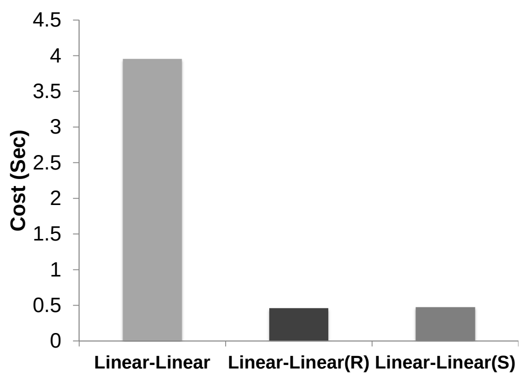
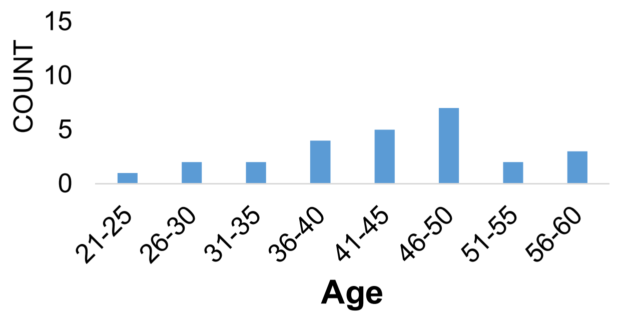


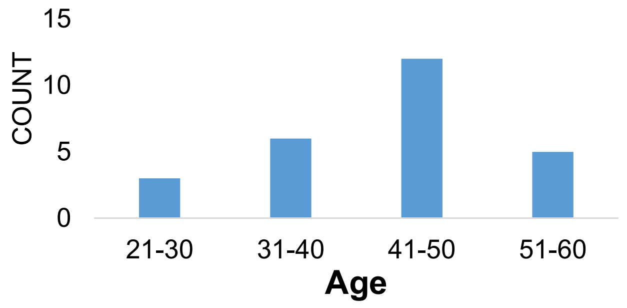
Figure 3.21: DIAB: Impact of View Refinement and Skipping Approximations on Cost

and *MuVE(G)-MuVE* by more than 50% compared to *Linear(G)-Linear* for high values of α_S. In Figure 3.20, we can see that the fidelity of *HC-Linear* is decreasing with increase in α_S while all other schemes have around 100% fidelity. This is because the geometric partitioning always includes views with small/medium number of bins (e.g., $2^0, 2^1, 2^2, 2^3$), which are typically the ones that provide high overall utility.

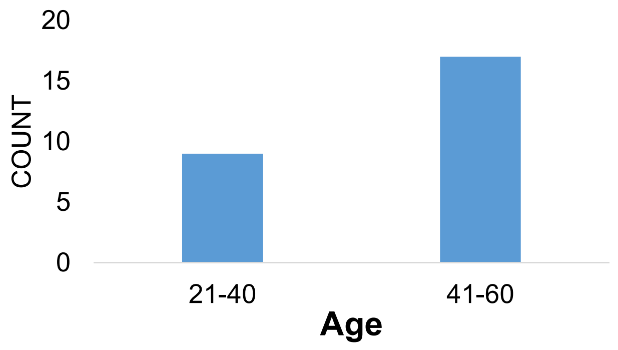
Approximation with View Refinement and Skipping (Figure 3.21): Figure 3.21 shows the cost of *Linear-Linear(R)* and *Linear-Linear(S)*, which employ approximations during the vertical search. As the figure shows, the cost of *Linear-Linear(S)* is lower than *Linear-Linear* because it reduces the search space by assuming that one bin size for a dimension would fit for all measures and functions. Meanwhile, *Linear-Linear(R)*, with default binning $def = 4$, offers the lowest cost as it vertically choses the top-k views in an initial pass and then the horizontal search is done only for those top-k views. In terms of fidelity, both *Linear-Linear(S)* and *Linear-Linear(R)* achieve around 95% fidelity.



(a) $V_{i,8}$



(b) $V_{i,4}$



(c) $V_{i,2}$

Figure 3.22: Generating $V_{i,2}$ by performing aggregation on $V_{i,4}$ or $V_{i,8}$

3.5 Materialized View Selection for Aggregate View Recommendation

3.5.1 Problem Definition

As mentioned in Section 4.3.1, the view recommendation process involves the generation of a huge number of the comparison and the target views. Particularly, these views are the result of executing their corresponding aggregate queries. Section 3.4.1 outlines how colossal the cost is for the binned view recommendation problem. However, we notice that for binned aggregate queries, the result of certain queries can be used to answer other queries.

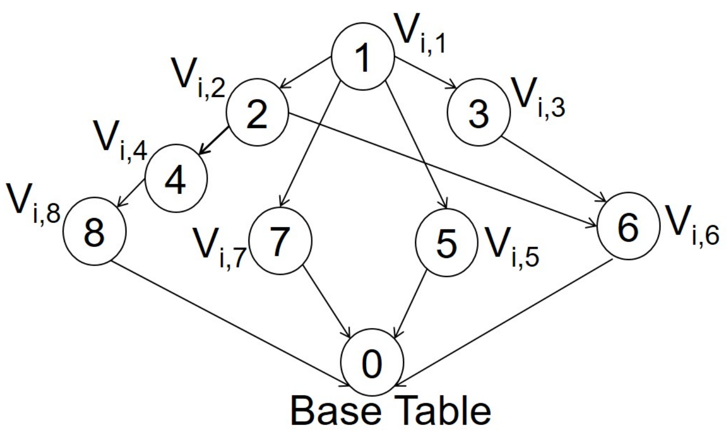


Figure 3.23: Lattice for View V_i with $B = 8$

Example 6. *For example, consider a table of employees, which has* Age *as a numerical dimension attribute. Particularly, one of the aggregate views on this attribute is count the number of employees grouped by* Age*. For this type of views, it is more meaningful if adjacent intervals are grouped together and shown in a summarized way. For example, Figure 3.22a shows the whole range grouped in 8 bins, while Figure 3.22b and 3.22c show range grouped in 4 bins and 2 bins respectively. It can be clearly seen that the view $V_{i,2}$ can be answered from views $V_{i,4}$ and $V_{i,8}$, by performing aggregation on these views instead of the base table.*

We term this relationship as dependency*. For instance, view $V_{i,2}$ depends on $V_{i,4}$, $V_{i,6}$ and $V_{i,8}$.*

Definition: View Dependency: a binned view $V_{i,b}$ depends on another binned view $V_{i,b'}$, if $V_{i,b}$ can be answered using $V_{i,b'}$, where b' is a multiple of b i.e., $b' = xb$.

For any non-binned view V_i, all the possible binned views $V_{i,b}$ can be directly generated from the base table. Therefore, every $V_{i,b}$ at least depends on the base table, and at most depends on $\frac{B}{b} - 1$ other views $V_{i,b'}$. The dependency relationship between the candidates can be represented by a lattice. Figure 3.23 shows the lattice for a particular non-binned view V_i that can have a maximum of 8 bins. Each node in the lattice represents a binned view, e.g. node 5 is binned view $V_{i,5}$, while node 0 represents the base table. A view can be generated using any of its ancestors in the lattice. For instance, the ancestors of node 3 (i.e.,$V_{i,3}$) are node 6 (i.e.,$V_{i,6}$) and node 0 (i.e., base table).

Every $V_{i,b'}$ is a candidate view that can be reused to generate some other views. Specifically, $V_{i,b}$ can be cached in the memory or stored on the disk for later reuse. However, because of the limited memory it is practical to store the view on the disk. Therefore, we propose to materialize the views that are later required to be reused. For instance, in Figure 3.23, every view that is an ancestor of at least one other view is a candidate view to be materialized. A key problem is how to decide which views should be reused? The three possible options are:

1. *Reuse nothing*: This is the baseline case in which all the queries are answered from the base table. Consequently, this would incur the query processing time for each binned view from scratch.

2. *Reuse the whole lattice*: In this case all views should be materialized. This would reduce the query processing time of each binned view but the overall execution time of the solution will increase because it would include the additional cost of materializing the views.

3. *Reuse a set of views*: Choose an optimal set of views $\mathbb{T}$ to reuse and materialize them. This will incur the cost of materialization but reduce the overall cost of the solution because a number of queries will be answered from the materialized views instead of the base table.

The best option is to reuse a set of views, which has a possibility of reducing the overall cost. However, a cost benefit analysis between answering the views directly from the base tables vs. materializing the views and answering some views from those materialized ones is required.

View Processing Cost

Recall from Section 3.4.1, the total cost incurred in processing a binned view $V_{i,b}$ is the sum of the costs of executing target and comparison queries, calculating deviation and accuracy (Eq 3.7). However, we note that the cost of computing deviation and accuracy is negligible as compared to query execution cost, as it involves no I/O operations. Furthermore, for simplicity in the next sections we assume $C(V_{i,b}) = C_t(V_{i,b}) + C_c(V_{i,b})$. Therefore, Eq 3.7 is reduced to:

$$C(V_i) = \sum_{b=1}^{B} C(V_{i,b}) \tag{3.11}$$

The goal of this study is to propose schemes that reduce the cost $C_t(V_{i,b})$ and $C_c(V_{i,b})$, which will consequently reduce the overall cost C of the solution. For that purpose, let $C_b(V_{i,b})$ be the cost of answering a binned view $V_{i,b}$ from the base table. Then in Eq 3.11, the cost of finding the top-1 binned view ($C(V_i)$), for the non-binned view V_i, can be rewritten as:

$$C(V_i) = \sum_{b=2}^{\frac{L}{w}} C_b(V_{i,b}) \tag{3.12}$$

Notice $C(V_i)$ actually specifies the cost for option 1, where nothing is reused. For the other options, where reuse is involved, let $C_m(V_{i,b})$ be the cost of answering $V_{i,b}$ from a materialized view. Additionally, let the views be divided into two sets: 1) *Dependent Set*: the views that can be answered from $\mathbb{T}$ belong to the dependent set $\mathbb{P}$, and 2) *Independent Set*: the views that cannot be answered from $\mathbb{T}$ belong to the independent set $\mathbb{I}$. Particularly, the views in $\mathbb{I}$ need to be answered from the base table. Let the cost of materializing a view $V_{i,b'}$ is $C_M(V_{i,b'})$, then Eq 3.13 specifies the cost for option 3:

$$C(V_i) = \sum_{V_{i,b} \varepsilon \mathbb{P}} C_m(V_{i,b}) + \sum_{V_{i,b} \varepsilon \mathbb{T}} C_M(V_{i,b}) + \sum_{V_{i,b} \varepsilon \mathbb{I}} C_b(V_{i,b}) \tag{3.13}$$

Definition: Materialized View Selection for View Recommendation: Given all the binned views $V_{i,b}$ for a non-binned view V_i, find a set $\mathbb{T}$ of views to materialize, which minimize the cost $C(V_i)$ of finding the top-1 binned view.

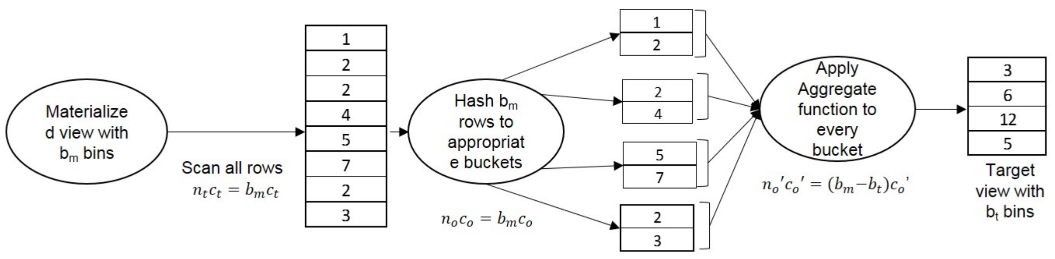

Materialize d view with b_m bins
Scan all rows
$n_t c_t = b_m c_t$
1
2
2
4
5
7
2
3
Hash b_m rows to appropriat e buckets
$n_o c_o = b_m c_o$
1
2
2
4
5
7
2
3
Apply Aggregate function to every bucket
$n_o' c_o' = (b_m - b_t) c_o'$
3
6
12
5
Target view with b_t bins

Figure 3.24: Example of Cost Model for HashAggregate Operator Where $b_m = 8$ and $b = 4$

In the next sections we present our proposed schemes that adapt and extend algorithms of materialized view selection towards efficiently solving the aggregate view recommendation problem.

3.5.2 mView: Greedy Approach

As explained int Section 3.3.3, the large number of possible binned views, makes the problem of finding the optimal binning for a certain view V_i highly challenging. An exhaustive brute force strategy is that given a certain non-binned view V_i, all of its binned views are generated and the utility of each of those views is evaluated. Consequently, the value of b that results in the highest utility is selected as the binning option for view V_i. However, this involves massive cost of processing all possible binned views.

In this work, we propose a novel technique *mView*, which instead of answering each query related to a view from scratch, reuses results from the already executed queries through view materialization. Particularly, *mView* maintains two sets of views; 1) $\mathbb{T}$: the views that are finalized to be materialized, 2) *Cand*: set of candidate views that can be added to $\mathbb{T}$ and consequently get materialized. The proposed technique *mView* adapts a greedy approach to determine $\mathbb{T}$ for materialization. Initially, *Cand* and $\mathbb{T}$ are empty. Then for a non-binned view V_i, a lattice as shown in Figure 3.23 is constructed, using an adjacency list after identifying dependencies among the views. The search for the top-1 binned view starts from the binned view $V_{i,b}$ where $b = 1$. All of the views that are ancestors of $V_{i,b}$ in the lattice are added to the set *Cand*. Next, the benefit of materializing each view in *Cand* is computed.

We study in detail how to compute the benefit of materializing a view in the next paragraph. After benefit calculation, from the set *Cand*, a view V_m, which provides the maximum benefit is selected. V_m is added in $\mathbb{T}$ if it is not already in $\mathbb{T}$. Consequently, V_m is materialized and $V_{i,b}$ is generated from V_m. In next iteration *Cand* is set to empty again and the ancestors of the next binned view are added to *Cand*. This process goes on until all of the $V_{i,b}$ have been generated. Clearly, for this technique to work efficiently, a cost model is required to estimate the benefit of materializing views without actual materialization. Therefore, next we define that cost and benefit model.

Cost Benefit Analysis

As mentioned earlier, to decide which views are the best candidates for materialization, the cost and benefit of materialization needs to be analyzed. Specifically, we use processing time as our cost metric

to measure performance of the schemes. In the linear cost model, the time to answer a query is taken to be equal to the space occupied by the underlying data from which the query is answered [79,80]. In this work, the same model is adopted with some modifications. Assume that the time to answer the aggregate query Q is related to two factors; 1) the number of tuples of the underlying view from which Q is answered, which is actually the number of bins of the ancestor view, and 2) the amount of aggregation required to answer Q. Normally, a relational DBMS uses HashAggregate as query execution plan for group-by queries. Particularly, in this study PostgreSQL is used as backend database, which uses HashAggregate as query execution plan for group-by queries. Hence, the cost model used by the query optimizer, particularly PostgreSQL consists of a vector of five parameters to predict the query execution time [81]; 1) Sequential page cost (c_s), 2) Random page cost (c_r), 3) CPU tuple cost (c_t), 4) CPU index tuple cost (c_i), and 5) CPU operator cost (c_o). The cost C_{HA} of the HashAggregate operator in a query plan is then computed by a linear combination of c_s, c_r, c_t, c_i, and c_o:
$C_{HA} = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o$
Where the values $n = (n_s, n_r, n_t, n_i, n_o)^{\mathbb{T}}$ represent the number of pages sequentially scanned, the number of pages randomly accessed, and so forth, during the execution.

Generally, for estimating cost of an operator, the values in vector n are estimated. However, in our case, the already known number of rows of a materialized view (i.e, number of bins of that view) and target view can be used for vector n. For instance, Figure 3.24 shows the steps of the HashAggregate operator for generating a view with 4 bins from a view with 8 bins, and the cost incurred. Specifically, The operation of generating a view with b bins from a view with b_m bins has the following parameters:

- $n_s c_s$ & $n_r c_r$: c_s and c_r are the I/O costs to sequentially access a page and randomly access a page, while n_s and n_r are the number of sequentially and randomly accessed pages respectively. Generally, size of a page is $8KB$. Consequently, n_s depends on the page size, size of each row (let it be r), and the number of rows read, which is equal to the number of bins of the materialized view, i.e., $n_s = \frac{8KB}{r \times b_m}$. Furthermore, c_s and c_r depends on whether the data is fetched from the disk or it is already in cache. Particularly, this cost is negligible for the later case and that is the case in our model.

- $n_t c_t$: c_t is the cost of scanning each row and n_t is the number of rows scanned, which is equal to the number of bins of the materialized view, i.e., $n_t = b_m$.

- $n_i c_i$: c_i is the cost to place the row in a bucket (bin) using hashing and n_i is the number of rows hashed, which is equal to the number of bins of the materialized view, i.e., $n_i = b_m$.

- $n_o c_o$: c_o is the cost to perform aggregate operation such as sum, count etc., and n_o is the number of aggregate operations performed. If $V_{i,b}$ is answered from V_{i,b_m}, then there are b buckets and each bucket will require $\frac{b_m}{b} - 1$ aggregate operations, i.e., $n_o = b(\frac{b_m}{b} - 1) = b_m - b$.

Therefore, the cost of HashAggregate operator C_{HA} is:
$C_{HA} = n_t c_t + n_i c_i + n_o c_o$

$C_{HA} = b_m c_t + b_m c_i + (b_m - b)c_o$

$C_{HA} = b_m(c_t + c_i + c_o) + b(-c_o)$

The costs c_t, c_i, and c_o remain same for all queries. therefore, we replace them with simple constants c and c' such that: $c = c_t + c_i + c_o$ and $c' = -c_o$. Hence,

$C_{HA} = b_m \times c + b \times c'$

Therefore, the cost of generating $V_{i,b}$ from materialized view V_{i,b_m} is:

$$C_m(v_{i,b}) = b_m \times c + b \times c' \tag{3.14}$$

Where c and c' are learnt through multi-variable linear regression. Consequently, the benefit of materializing a view V_{i,b_m} is computed by adding up the savings in the query processing cost for each dependent view $V_{i,b}$ over answering $V_{i,b}$ from the base table and subtracting the cost of materialization of V_{i,b_m}.

$$\mathbb{B}(V_{i,b_m}) = \sum_{V_{i,b} \in \mathbb{P}} [(C_b(V_{i,b}) - C_m(V_{i,b}))] - C_M(V_{i,b_m}) \tag{3.15}$$

In this section, we listed the details of our proposed technique *mView* for the exhaustive search, which is also called *Linear* search. When this scheme is applied to a non-binned view V_i, it results in a top-1 binned view, this is termed has horizontal search. Furthermore, applying this to every non-binned view, their corresponding top-1 binned views are identified and from there top-k views can be easily recommended, this is termed as vertical search. In our experiments, we differentiate between horizontal and vertical search and the scheme applied to each direction.

3.5.3 Materialized views with MuVE

In [1, 2], we argue that the deviation based utility metric falls short in completely capturing the requirements of numerical dimensions. Hence, a hybrid multi-objective utility function was introduced, which captures the impact of numerical dimension attributes in terms of generating visualizations that have: 1) interestingness ($D(V_{i,b})$): measured using the deviation-based metric, 2) usability ($S(V_{i,b})$): quantified via the relative bin width metric, and 3) accuracy ($A(V_{i,b})$): measured in terms of Sum Squared Error (SSE). The proposed multi-objective utility function, was defined as follows:

$$U(V_{i,b}) = \alpha_D \times D(V_{i,b}) + \alpha_A \times A(V_{i,b}) + \alpha_S \times S(V_{i,b}) \tag{3.16}$$

Parameters α_D, α_A and α_S specify the weights assigned to each objective, such that $\alpha_D + \alpha_A + \alpha_S = 1$. Furthermore, to efficiently navigate the prohibitively large search space *MuVE* scheme was proposed, which used an incremental evaluation of the multi-objective utility function, where different objectives were computed progressively. In this section, we discuss how to achieve benefits of both the schemes, *mView* and *MuVE*.

Selecting $\mathbb{T}$ while using *MuVE* as search strategy is non-trivial, because of the trade-off between *MuVE* and *mView*. In the *MuVE* scheme, the benefit of cost savings comes from the pruning of many views and utility evaluations. A blind application of greedy view materialization, as in *mView*, may result in materialization of views that gets pruned because of the *MuVE's* pruning scheme. The idea here is to estimate which views *MuVE* will eliminate and exclude those views from the set of candidate views to materialize. To address this issue, we introduce a penalty metric, which is added to the benefit function. Therefore, a candidates view V_{i,b_m}, which has high certainty (represented as $CE(V_{i,b_m})$) of getting pruned by *MuVE* gets a high reduction in its benefit of materialization. Particularly, a view gets pruned due to either of the two factors; 1) short circuit of deviation objective, the certainty of this pruning is represented as $CE_D(V_{i,b_m})$, and 2) early termination, certainty of getting early terminated is represented as $CE_E(V_{i,b_m})$. The certainty factor $CE(V_{i,b_m})$ is the sum of the certainty of pruning deviation evaluation ($CE_D(V_{i,b_m})$) and certainty of getting early terminated ($CE_E(V_{i,b_m})$).

$$CE(V_{i,b_m}) = CE_D(V_{i,b_m}) + CE_E(V_{i,b_m})$$

Therefore, the benefit of materializing a view in Eq 3.15 is updated as:

$$\mathbb{B}(V_{i,b_m}) = \sum_{V_{i,b} \in \mathbb{P}} [C_b(V_{i,b}) - C_m(V_{i,b})] - [CE(V_{i,b_m}) \times C_M(V_{i,b_m})] \tag{3.17}$$

The certainty of pruning deviation computation depends on the ratio of α_A and α_D. *MuVE* uses a priority function to determine which objective to evaluate first, in other words *MuVE* tries to prune the objective, which is not evaluated first. According to that function if α_A is greater than α_D there is a chance of pruning the deviation objective. We are interested in pruning deviation evaluation as it is the only objective that involves execution queries for target and comparison views.

$$CE_D(V_{i,b}) = \left\{ \begin{matrix} 0\, for\ \frac{\alpha_A}{\alpha_D} < 1 \\ \frac{\alpha_A}{\alpha_D} \times 10\, for\ \frac{\alpha_A}{\alpha_D} \geq 1 \end{matrix} \right\} \tag{3.18}$$

The certainty of early termination depends on α_S and b, higher value of α_S or b means the chance of getting early termination is high.

$$CE_E(V_{i,b}) = \left\{ \begin{matrix} 0\, for\ \alpha_S < 0.5 \\ \alpha_S \times \frac{b}{L}\, for\ \alpha_S \geq 0.5 \end{matrix} \right\} \tag{3.19}$$

3.5.4 Experimental Testbed

We perform extensive experimental evaluation to measure the efficiency of top-k view recommendation strategies presented in this section. Here, we present the different parameters and settings used in our experimental evaluation.

Setup: We built a platform for recommending visualizations, which extends the SeeDB codebase [6] to support view materialization based schemes presented. Our experiments are performed on a Corei7 machine with 16GB of RAM. The platform is implemented in Java and PostgreSQL is used as the backend DBMS.

Schemes: We investigate the performance of the different combinations of the vertical and horizontal search strategies presented in [1] with *mView* proposed in this thesis. Our naming convention for

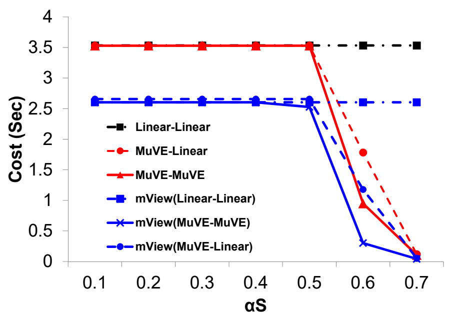


Figure 3.25: Impact of α_A and α_S on cost, while $\alpha_D = 0.2$

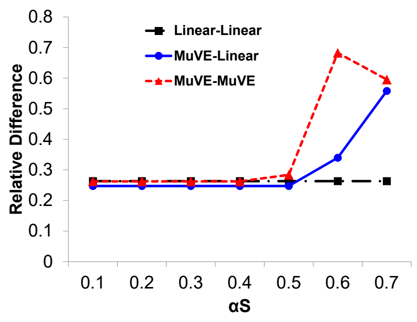


Figure 3.26: Impact of α_A and α_S on relative difference, while $\alpha_D = 0.2$

those combinations is represented as: *SearchH-SearchV*, where *SearchH* denotes the search strategy employed for horizontal search, whereas *SearchV* is the one for the vertical search. This leads to the following combinations: *Linear-Linear*, *MuVE-Linear*, and *MuVE-MuVE* as baseline schemes and *mView(Linear-Linear)*, *mView(MuVE-Linear)*, and *mView(MuVE-MuVE)* as proposed schemes.

Data Analysis: As in [6], we assume a data exploration setting in which a multi-dimensional dataset of diabetic patients[1] is analyzed. The DIAB dataset has 9 attributes and 768 tuples. The independent numeric attributes of the dataset are used as dimensions (e.g., age, BMI, etc.), whereas the observation attributes are used as measures (insulin level, glucose concentration, etc.). In our default setting, we select 3 dimensions, 3 measures, and 3 aggregate functions, which results in a maximum of 2961 possible views. In the analysis, all the α values are in the range $[0-1]$, where $\alpha_D + \alpha_A + \alpha_S = 1$. In the default setting, $\alpha_D = 0.2$, $\alpha_A = 0.2$, $\alpha_S = 0.6$, $k = 5$, and euclidean distance is used for measuring deviation, unless specified otherwise.

Performance: We evaluate the efficiency and effectiveness of the different recommendations strategies in terms of two factors:1) *Cost:* As mentioned in Section 4.4, the cost of a strategy is the total cost incurred in processing all the candidate binned views. We use wall clock time to measure the different components included in that cost namely, query execution time of target and comparison views, deviation computation time, and accuracy evaluation time. 2) *Relative Difference:* The ratio between cost of baseline schemes and the *mView* based schemes, i.e., $\frac{Cost of baseline - Cost of mView Scheme}{Cost of baseline scheme}$. Each setting is executed 10 times and then average is taken as the cost incurred.

3.5.5 Experimental Evaluation

In the following experiments, we evaluate the performance of our technique *mView* under different parameter settings. As explained in Section **??** that *mView* scheme is used in combination with the baseline Linear scheme and optimized *MuVE* scheme. Additionally it was also mentioned that the blind materialization of views while using *MuVE* search strategy may not be the optimal solution. Therefore, for *mView(MuVE-MuVE)* and *mView(MuVE-Linear)* schemes an heuristic based method was proposed to predict the expected early termination and short circuit point. Figures 3.25 and 3.27

[1]https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

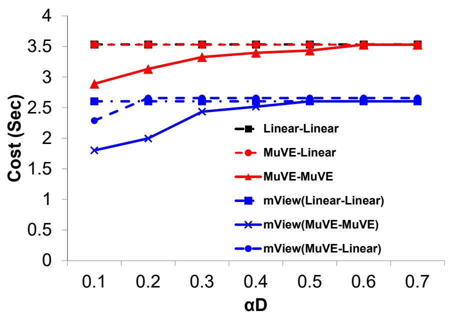


Figure 3.27: Impact of α_A and α_D on cost, while $\alpha_S = 0.2$

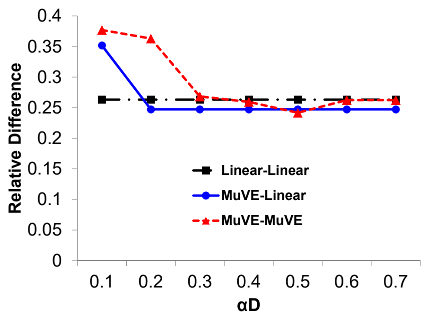


Figure 3.28: Impact of α_A and α_D on relative difference, while $\alpha_S = 0.2$

show the impact on cost, while Figures 3.26 and 3.28 quantifies the percentage improvement achieved in terms of relative difference using the view materialization scheme.

In Figures 3.25 and 3.26, α_D is set to constant 0.2 while α_A and α_S are changing. In particular, as shown in the figures, α_S is increased, while α_A is implicitly decreased and is easily computed as $\alpha_A = 1 - \alpha_D - \alpha_S$. Figure 3.25 shows that cost *mView(Linear-Linear)* is less than the baseline scheme *Linear-Linear*. This is because *mView* chooses such a set of views to materialize that saves aggregation time by generating them from the materialized views. Furthermore, Figure 3.26 shows *mView(Linear-Linear)* reduces the cost by almost 30% as compared to the *Linear-Linear* scheme. Figure 3.25 also shows that using our proposed heuristic in *mView(MuVE-MuVE)* and the incremental view materialization of *mView*, the cost is further reduced. This is due to the reason that we avoided the unnecessary materialization of views which are eventually pruned by *mView(MuVE-MuVE)*. Furthermore, Figure 3.26 shows *mView(MuVE-MuVE)* reduces the cost by almost 70% as compared to *MuVE-MuVE* at $\alpha_S = 0.6$.

In Figures 3.27 and 3.28, α_S is set to constant 0.2 while α_A and α_D are changing. Figure 3.27 clearly shows that *mView* based three schemes have less cost compared to the other three schemes. The difference in cost for the *mView(MuVE-MuVE)* scheme is more than 30% at $\alpha_D = 0.1$ as shown in Figure 3.28.

3.6 Summary

In this chapter we presented a novel utility function and a suite of search schemes for recommending top-k aggregate data visualizations. Our utility function recognizes the impact of numerical dimensions on visualization, which is captured by means of multiple objectives, namely: deviation, accuracy, and usability. Our proposed search schemes further incorporate that utility function for the purpose of recommending the top-k aggregate data visualizations. A key goal in the design of those search schemes is to efficiently prune the prohibitively large search space of possible data aggregations. That goal is reasonably achieved by our *MuVE* scheme, and is further improved by *uMuVE*, at the expense of a high memory usage. Accordingly, we presented *MuMuVE*, which provides a pruning power close to that of uMuVE, while keeping memory usage within predefined constrain. Moreover, we also presented another novel technique *mView* for recommending top-k binned aggregate data visualizations. The proposed scheme reuses the already executed views through materialization and answering the later queries from the materialized views. We have provided extensive experimental evaluation, which illustrate the benefits achieved by proposed schemes.

Chapter 4

Input Query Refinement for View Recommendation

4.1 Introduction

Visual data exploration is the rudiments of deriving insights from large datsets. Typically, it involves an analyst going through the following steps: 1) selecting a subset of data for analysis, 2) generating different visualizations of that analyzed data, and 3) sifting through those visualizations looking for the ones that reveal interesting insights. Based on the outcome of the last step, the analyst might have to refine their initial selection of data so that the newly analyzed subset would show more interesting insights. This is clearly an iterative process, in which each selection of data (i.e., input query) is a springboard to the next one. For this time-consuming process to be effective, a challenging combination of system and domain expertise is required.

Motivated by the need for an efficient and effective visual data exploration process, several solutions have been proposed towards automatically finding and recommending interesting data visualizations (i.e., steps 2 and 3 above) [1,6–9]. The main idea underlying those solutions is to automatically generate all possible views of data, and recommend the top-k interesting views, where an interestingness of a view is quantified according to some utility function. Recent work provides strong evidence that a deviation-based formulation of utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [1,6,9,15]. In particular, the deviation-based metric measures the distance between the probability distribution of a visualization over the analyzed dataset (i.e., target view) and that same visualization when generated from a reference dataset (i.e., comparison view), where the reference dataset is typically the entire database from which that analyzed data is extracted. The underlying premise is that a visualizations that results in a higher deviation is expected to reveal some interesting insights that are very particular to the analyzed dataset [1,6,9,15].

Existing solutions have been shown to be effective in recommending interesting views under the assumption that the analyst is precise in their selection of analyzed data. That is, the analyst is

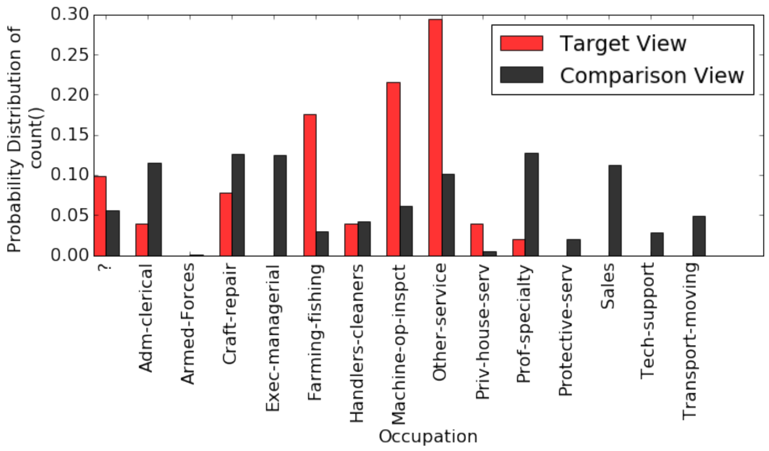


Figure 4.1: View from Refined Query Q_1 (Far from Q)
Target View $V_i(D_{Q_1})$: `SELECT Occupation COUNT(*) FROM C WHERE education` $\leq$ `1 GROUP BY Occupation`
Comp. View $V_i(D_B)$: `SELECT Occupation COUNT(*) FROM C GROUP BY Occupation`
Deviation = `0.2614147363382412`

able to formulate a well-defined input query that selects a subset of data, which contains interesting insights that can be revealed by the recommended visualizations. However, such assumption is clearly impractical and extremely limits the applicability of those solutions. In reality, it is typically a challenging task for an analyst to select a subset of data that has the potential of revealing interesting insights. Hence, it is a continuous process of trial and error, in which the analyst keeps refining their selection of data manually and iteratively until some interesting insights are revealed.

Therefore, in this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. That is, there is a need for solutions in which the two tasks of data selection as well as view recommendation are both automated and work together in synergy.

To further illustrate the need for such solution, we presented Ex. 2 in Chap 1. The example illustrated a clear need for a query refinement solution that is able to automatically modify the analyst's initial input query into a new query, which selects a subset of data that includes interesting insights. Those hidden insights are then easily revealed using existing solutions that are able to recommend interesting visualizations. To that end, one straightforward and simple approach would involve generating all the possible subsets of data by automatically refining all the predicates of the input query. In our example above, that would be equivalent to generating all refinements of the predicate `WHERE education` $\geq$ `12`. Consequently, for each subset of data selected by each query refinement, generate all possible aggregate views (i.e., visualizations). In addition to the obvious challenge of a prohibitively large search space of query refinements, that naive approach would also lead to visualizations that might appear to be visually interesting but they are useless from the analyst's perspective. To illustrate that limitation, consider the following example.

Example 7. *Now assume that the naive approach described above is applied to the input query Q*

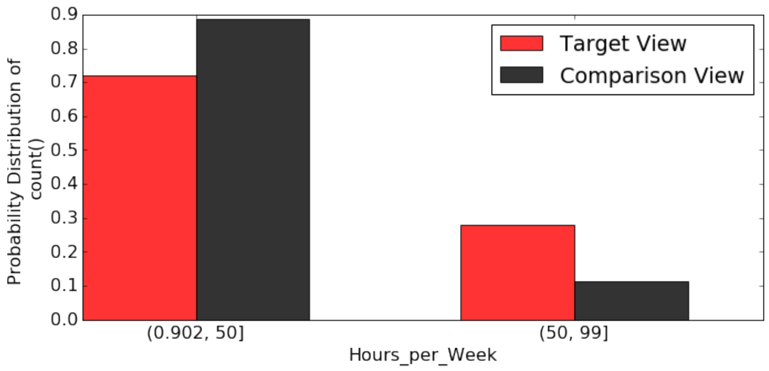


Figure 4.2: View from Refined Query Q_2 (Close to Q)
Target View $V_i(D_{Q_2})$**:** `SELECT Hours_per_Week COUNT(*) FROM C WHERE education` $\geq$ `16 GROUP BY Hours_per_Week`
Comp. View $V_i(D_B)$**:** `SELECT Hours_per_Week COUNT(*) FROM C GROUP BY Hours_per_Week`
Deviation = `0.16898935750541005`

specified in Example 2. That is, the all possible refinements of Q are generated so that to find and recommend visualizations that are more interesting than the one shown in Figure 1.4. Particularly, after generating all those possible refined queries with all the possible values for the predicate on the `education` *attribute, the recommended top visualization is shown in Figure 4.1. That visualization is recommended based on the following refined query* Q_1*:*

Q_1`:SELECT * FROM C WHERE education` $\leq$ `1,`

As Figure 4.1 shows, that visualization is based on plotting the probability distribution of the `occupation` *attribute for those who never went to school (i.e.,* `education` $\leq$ `1`*) vs. the population.*

Clearly, Figure 4.1 is visually interesting as the probability distribution of the target view is significantly different from the comparison view (i.e., deviation = 0.261414..).

However, there are two issues with that refinement, and in turn the recommended visualization: 1) similarity-oblivious: a blind automated refinement that is oblivious to the analyst's preferences might result in a refined query that is significantly dissimilar from the input query. For instance, in this example the analyst's intention is to analyze the subset of data for those who completed high school (i.e., `education` $\geq$ `12`), whereas the refined query selects for the analysis those who never went to school (i.e., `education` $\leq$ `1`). 2) statistical insignificance: as Figure 4.1 shows, the target view is missing a number of values for the `occupation` attribute, which indicates that the subset selected by the refined query Q_1 is possibly too small for analysis, and in turn statistically insignificant.

The two issues mentioned above highlight the need for automatic refinement solutions that are guided by the user's preference, which is precisely the focus of this work. In particular, we propose a novel scheme for automated query refinement for view recommendation in visual data exploration, called QuRVe. Before discussing the details of QuRVe in the next sections, we illustrate its benefits using following example:

Example 8. *Figure 4.2 shows the top view recommended by QuRVe based on the input query Q provided in Example 2. That view is generated based on automatically refining Q into the new modified* *and statistically significant query Q_2, which is specified as:*

Q_2:`SELECT * FROM C WHERE education` $\geq$ `16`

Notice that Q_2 is clearly more similar to the input query Q than the previously refined query Q_1 discussed in Example 7. Particularly, instead of analyzing the data for those who completed high school (i.e., `education` $\geq$ `12`*), the refined query Q_2 analyzes the data for those who completed a college degree (i.e.,* `education` $\geq$ `16`*). More importantly, the recommended view based on the refined Q_2 shows a uniquely interesting insight. Specifically, as Figure 4.2 shows, highly educated people tend to work more hours than the rest of the population. More precisely, it shows that only 13% of the population work more than 50 hours a week, whereas that percentage goes up to 30% for those have completed college.*

Based on the previous example our key goal in designing QuRVe is to recommend interesting visualizations, while at the same time achieving high effectiveness and efficiency. To ensure that desired effectiveness, we formulate the problem of refining query for recommending top-k aggregate visualization as a multi-objective optimization problem. Particularly, given an input query Q, the optimization objective is to find those top-k interesting visualizations from all possible refinements of the input query according to a similarity-aware utility function, subject to predefined constraints on statistical significance. Clearly, such formulation is challenged by the large number of possible refinements and corresponding visual representations generated per refinement. Hence, to achieve efficient recommendation, QuRVe introduces novel search algorithms that are particularly optimized to leverage the specific features of the problem for pruning the large search space.

4.2 Related Work

4.2.1 Query Refinement

Our work stands out from existing query refinement techniques because it addresses a novel problem of automatic query refinement for statistically significant visual recommendation. The existing work of query refinement in the domain of data exploration can be divided into three categories. Firstly, the refinement of queries for satisfying cardinality constraints. Secondly, exploring predicates to answer why questions and to explain outliers. Thirdly, refining queries for general aggregate results. [82] is the seminal work on query refinement to satisfy cardinality constraints on the query result. This cardinality constrained based refinement has applications in the many/few answers problems. This work addresses a relaxed problem compared to us, as only partial combinations are generated due to interactivity with the user. In comparison, core of our work is around the automation objective, however we also combine the user preference through weighted similarity metric and other control parameters of the proposed schemes. Scorpion [83] similar to us, is another system to analyze large datasets through aggregates. However, while our system refine predicates to find interesting aggregate views, scorpion takes a set of outlier points in an aggregate query result as input and finds predicate that explain the outliers. Aggregate operator's properties are used to derive the search and prune the predicate search space.

SAQR [84] proposed scheme for similarity-aware refinement of aggregate queries, which satisfy the aggregate and similarity constraints imposed on the refined query and maximize its overall utility. The recent research with similarities to our work is [85], however their goal is to generate set of alternative queries to meet the constraint on similarity to the original query and the aggregate constraint. While, our goal is to generate alternate queries to maximize utility based on similarity to the original query and deviation from the reference dataset. The proposed partitioning based optimization to speed up query processing because of overlap between refined queries is orthogonal to our work, as we also have these overlapping queries.

4.2.2 Hypothesis Testing

Hypothesis testing is a well studies area in the domain of statistics. However, very little work has been done in the domain of data exploration in terms of determining statistical significance of explored data. Towards exploratory hypothesis testing and analysis [86], was one of the initial attempts to connect hypothesis testing to database domain. They find sub-populations for comparison, using frequent pattern mining techniques and then perform significance tests. The hypothesis that are found statistically significant are further analyzed. Another work related to statistically significant data driven discoveries is [87]. In this work a framework is proposed that allows the user to query for statistically significant relationships between data sets, while our work focuses on finding statistically significant visualizations within one dataset. Additionally, restricted monte carlo tests for correlations of data sets are developed. The initial experiments indicate that the significance tests can help the data discovery process. Similar to our work is [55], which automatically creates hypothesis based on aggregate queries and perform significance testing based on chi-square tests.

4.3 Preliminaries

In this section, we first describe the basics of view recommendation, followed by automatic query refinement in general. We also describe testing statistical significance and estimating sample sizes based on hypothesis testing for view recommendation.

4.3.1 View Recommendation

Similar to the recent data visualization platforms [1, 6–8, 17, 64], we are given a multi-dimensional dataset $D_B(\mathbb{A},\mathbb{M})$, where $\mathbb{A}$ is the set of dimension attributes and $\mathbb{M}$ is the set of measure attributes. Additionally, $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$. In a typical visual data exploration session the user chooses a subset D_Q of the dataset D_B by issuing an input query Q. For instance, consider the following query Q:

Q: SELECT * FROM D_B WHERE T;

In Q, T specifies a combination of predicates, which selects D_Q for visual analysis (e.g., `education` $\geq$ `12` in Example 2). A visual representation of Q is basically the process of generating an aggregate

Symbol	Description
D_B	Database
D_Q	Subset of data selected by Q
Q	Input query
T	Predicates in input query
Q_j	A refined query
$\mathbb{Q}$	Set of refined queries
$\mathbb{A}$	Set of dimension attribute
$\mathbb{M}$	Set of measure attribute
$\mathbb{F}$	Set of aggregate function
$\mathbb{P}$	Set of attributes for predicates
$V_i(D_Q)$	i^{th} Aggregate view on D_Q
$V_{i,Qj}$	i^{th} Aggregate view on refined query Q_j
N	Total number of views
$D(V_i)$	Deviation of a view V_i
$S(Q,Q_j)$	Similarity between Q and Q_j
D_M	Theoretically maximum deviation of any view
D_u	Upper bound on deviation of any views

Table 4.1: Summary of symbols

view V_i of its result (i.e., D_Q), which is then plotted using some visualization methods such as bar charts, scatter plots, etc. Therefore, an aggregate view V_i over D_Q is represented by a tuple (A,M,F,b) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, $F \in \mathbb{F}$ and b is the number of bins in case A is numeric. That is, D_Q is grouped by dimension attribute A and aggregated by function F on measure attribute M. For instance, (Hours_per_Week, *,COUNT,2) represents the aggregate view shown in Fig. 1.4.

Manually finding interesting views of data is a time-consuming task. Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on deviation based metric (e.g., [1,6–8,17]). In particular, it measures the deviation between the aggregate view V_i generated from the subset data D_Q vs. that generated from the entire database D_B, where $V_i(D_Q)$ is denoted as *target* view, whereas $V_i(D_B)$ is denoted as *comparison* view. To ensure that all views have the same scale, each target view $V_i(D_Q)$ and comparison view $V_i(D_B)$ is normalized into a *probability distribution* $P[V_i(D_Q)]$ and $P[V_i(D_B)]$ and it is bounded by the maximum deviation value D_M. Accordingly, the deviation $D(V_i)$, provided by a view V_i, is defined as the normalized distance between those two probability distributions.

$$D(V_i) = \frac{dist(P[V_i(D_Q)],P[V_i(D_B)])}{D_M} \tag{4.1}$$

Consequently, the deviation $D(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) [1,2,6,15]. Hence, the number of possible views to be constructed is $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$, which is clearly inefficient for a large multi-dimensional dataset.

As explained earlier, the input query provided by the user is the cornerstone for recommending interesting views. However, formulating a well-defined input query that selects a subset of data, which contains interesting views is a non-trivial task. Therefore, in this work we argue that in addition to existing visualization recommendation solutions, there is an equal need for query refinement solutions, that automatically refine input query to select subsets of data that would reveal interesting visualizations. In the next section we discuss the preliminaries of refining a query.

4.3.2 Query Refinement

Automatic query refinement is a widely used technique for DBMS testing, information retrieval and data exploration. In a nutshell, in this technique the user provides an initial query and then it is progressively refined to meet a particular objective. In the context of data exploration and aggregate queries, query refinement has been used to automatically recommend queries for satisfying cardinality and aggregate constraints [82, 85], explaining outliers [83] and answering why not questions [88].

In this work, we propose to automatically refine input query for the objective of view recommendation. Particularly, as mentioned in Section 4.3.1, the user provides an input query Q, in which T specifies a conjunction of predicates. The input query is progressively refined by automatically enumerating all combinations of predicates for the objective of generating interesting views.

Particularly, we consider queries having numeric selection predicates with range $(<, \leq, >, \geq)$ operators. These predicates are defined on a set of numeric dimension attributes denoted as $\mathbb{P}$. The number of predicates is p, such that $|\mathbb{P}| = p$. Each of this range predicate is in the form $l_i \leq P_i \leq u_i$ where $P_i \in \mathbb{P}$ and l_i and u_i are the lower and upper limits of query Q along predicate P_i. The domain of predicate P_i is limited by a Lower bound L_i and upper bound U_i. A refined query Q_j for an input query Q is generated by modifying the lower and/or upper limits for some of the predicates in Q. That is, for a predicate $l_i \leq P_i \leq u_i$ in query Q, a refined predicate in Q_j takes the form $l_i' \leq P_i \leq u_i'$. The predicate that is not included in T is equivalent to $L_i \leq P_i \leq U_i$. Similar to [82, 84], we convert the range predicated into two single sided predicates. Therefore, $l_i \leq P_i \leq u_i$ is converted to two predicates: $P_i \leq u_i \bigwedge -P_i \leq -l_i$. This allows refinement of one or both sides of the range predicates and this results in the total number of single sided predicates to be $2p$. The refinement can be in one of the two directions: i) *contracting* i.e. decreasing the value of the predicate and ii) *expanding* i.e. increasing the value of the predicate. The set of all of the refined queries is denoted as $\mathbb{Q}$. The number of all possible refinements is exponential in p and forms a combinatorial search space. For instance, if predicate P_i is discrete and refinements are in step of 1.0, then number of all possible refinements are $n_i = \frac{a_i(a_i+1)}{2}$ where $a_i = U_i - L_i$. For p such predicates the combinations of all possible refinements are $n_1 \times n_2 \times ......n_p \approx n^p$. In other words size of the set $\mathbb{Q}$ is approximately n^p (i.e., $|\mathbb{Q}| \approx n^p$). Consider the following example to clearly understand the predicate specification and refinement.

Example: Consider the query Q in Example 2, in Q only $l_i = 12$ is defined explicitly, however $u_i = U_i = 16$ is automatically added and Q is completely defined as follows:

Q:SELECT * FROM C WHERE 12 ≤ education ≤ 16

After converting to single sided predicated it would become

```
education ≤ 16 and -education ≤ -12
```

A refined query Q_j is obtained by expanding or contraction one or more predicates $P_i \in T$ to P'_i, this makes Q_j different from the input query Q. A refined query that is significantly dissimilar from the input query, results in loss of user preference. Therefore, to quantify the change that has been made to Q to get the refined query Q_j, we define a similarity measure $S(Q,Q_j)$ in terms of the distance of Q_j from Q i.e., $s(Q,Q_j)$.

$$S(Q,Q_j) = 1 - s(Q,Q_j) \tag{4.2}$$

In Section 4.5, we define the distance $s(Q,Q_j)$ to quantify the change made to the input query to get the refined query.

4.3.3 Hypothesis Testing

In visual exploration sometimes the visually observed difference may not actually be statistically significant, specially in the case of view recommendation, which involves summary data based visualizations and their ranking [55, 89, 90]. Particularly, with our recommendation tool [1, 2], we made the following observations:

- Some of the recommended top-k target views have very few underlying tuples, which lead to higher deviation values. Consequently, such views receive higher rank despite of the lack of real insight, for instance, the target view shown in Fig 4.1 of Example 7.
- Sometimes the subset under analysis has only low deviation views, for instance, the case discussed in Example 2. Furthermore, in such situation there are many views with very similar deviation values. This actually means that the difference observed is not statistically significant and the top-k recommendation must consider additional criteria along with the deviation value. For instance, the target view shown in Fig 1.4 of Example 2 is not statistically significant.

To determine whether the observed difference is statistically significant, we employ the widely used approach of hypothesis testing. Hypothesis testing determines if there is enough evidence for inferring that a difference exists between two compared samples or between a sample and population. A difference is called statistically significant if it is unlikely to have occurred by chance [89]. Hypothesis testing involves testing a null hypothesis by comparing it with an alternate hypothesis. The hypothesis to be tested is called the *null hypothesis*, denoted as H_0. The null hypothesis states that there is no difference between the population and the sample data. The null hypothesis is tested against an *alternate hypothesis*, denoted as H_1, which is what we have observed in the sample data. For instance, in Fig 1.4 of Example 2, the hypothesis is that the high school graduates work different number of hours per week (Hours worked is divided into two categories) as compared to the population, and this becomes the alternate hypothesis. The corresponding null hypothesis is that no such difference exits. Likewise all the views from refined queries become the alternate hypothesis.

Depending on the nature of the statistical test and underlying hypothesis, different null hypothesis statistical tests have been developed, e.g., z-test and t-test for normal distribution. Moreover, these tests assume that the data arise from a distribution described by parameters, for instance a normal distribution is described with mean and variance. The choice of statistical test is based on the domain and data under analysis .

Furthermore, after stating null and alternative hypothesis, the chosen statistical test returns the *p-value*. The p-value is the probability of obtaining a statistic at least as extreme as the one that was actually observed, given the null hypotheses is true. Accordingly, p-value for each hypothesis (i.e., $pvalue(V_i)$) is computed. Smaller p-value means it is very unlikely that the observed difference is by chance and thus it is statistically significant. The p-value is compared against a priori chosen *significance level* α. The conventionally used significance level is 0.05. if $pvalue(V_i) \leq \alpha$, then the null hypothesis H_0 must be rejected, which means the V_i is statistically significant. Due to the nature of the statistical test involved, the acceptance or rejection of H_0 can never be free of error. If the test incorrectly rejects or accepts H_0, then an error has occurred. There can be following two types of errors in hypothesis testing:

1. If H_0 is rejected, while it was true, it is called *Type-I error*.
2. If H_0 is accepted, while H_1 was true, it is called *Type-II error*.

Type-II error is critical in our case because we do not want to reject views that might be interesting. The probability of Type-II error is specified by user specified parameter β, which normally has a value between 0.10 and 0.20. An alternate term is power, which is the probability of rejecting a false null hypothesis, therefore, $power = 1 - \beta$. A priori power analysis determines the minimum sample size to obtain required power. By setting an expected size (ω), significance level (α), and power level (β), the sample size necessary to meet this specification can be determined [91].

Although we can control our search by giving more weight to similarity, there can still be a situation where the number of tuples is small. Consequently, it will lead to a target view having empty groups. Hence, it will result in high deviation from the comparison view. This leads to false discoveries as the high deviation is because of insufficient sample size, and not due to the different distribution of target view from comparison view. Therefore, we perform power analysis, estimate the minimum sample size required to achieve the specified power, and employ it as a constraint in our problem definition. Consequently, the refined subsets that satisfy the minimum sample constraint participate in our search of top-k views.

4.4 Query Refinement for View Recommendation

4.4.1 View Recommendation with Query Refinement

In this paper, we mainly focus on recommending the top-k aggregate views generated on the input query Q and all refined queries $Q_j \in \mathbb{Q}$. In Section 4.3.1, we defined aggregate view V_i, on user selected subset of data D_Q (recall D_Q is selected by input query Q). Accordingly, the Eq. 4.1 defined the deviation of an aggregate view $D(V_i)$, by calculating normalized distance between probability distributions of $V_i(D_Q)$ and $V_i(D_B)$. However, after refining the input query and generating all $Q_j \in \mathbb{Q}$, we have views V_i from input query as well as from refined queries. Therefore, to identify origin of a view V_i, we denote it as V_{i,Q_j}, which means i^{th} view of the query Q_j. Accordingly, the Eq. 4.1 can be modified to define the deviation $D(V_{i,Q_j})$, provided by a view V_{i,Q_j}, as:

$$D(V_{i,Q_j}) = \frac{dist(P[V_i(D_{Q_j})], P[V_i(D_B)])}{D_M} \tag{4.3}$$

In next section, we formally define the problem of query refinement for view recommendation.

4.4.2 Problem Statement

In a nutshell, the goal of this work is to recommend the top-k bar chart visualizations of the results of query Q and all refined queries $Q_j \in \mathbb{Q}$ according to some utility function. When the visualizations are only of query Q such goal simply boils down to recommending the top-k interesting views based on the deviation metric, as described in Section 4.3.1. However, that simple notion of utility falls short in capturing the impact of refinement on input query. In particular, the automatic refinement introduces additional factors that impact the interestingness and in turn utility of the recommended views. In our proposed schemes, we employ a weighted multi objective utility function and some constraints to integrate such factors, namely:

1. *Interestingness:* Is the ability of a view to reveal some interesting insights about the data, which is measured using the deviation-based metric $D(V_{i,Q_j})$ (Eq. 4.3).

2. *Similarity:* Is the similarity of the underlying refine query Q_j of the view V_{i,Q_j}, with the input query Q, it is measured in terms of the similarity metric $S(Q,Q_j)$ (Eq. 4.2).

3. *Statistical Significance:* Is the ability of the query Q_j and the view V_{i,Q_j} to generate statistically significant result, which is measured as a constraint by $power(Q_j)$ to check the size of the subset selected by query Q_j and $pvalue(V_{i,Q_j})$ to check significance of the view V_{i,Q_j}.

To capture the factors and constraints mentioned above, we first employ a weighted multi-objective utility function, which is defined as follows

$$U(V_{i,Q_j}) = \alpha_S \times S(Q,Q_j) + \alpha_D \times D(V_{i,Q_j}) \tag{4.4}$$

where $S(Q,Q_j)$ is the similarity between input query Q and refined query Q_j of the view V_{i,Q_j}, and $D(V_{i,Q_j})$ is the normalized deviation of view V_{i,Q_j} from the overall data.

Parameters α_S and α_D specify the weights assigned to each objective in our hybrid utility function, such that $\alpha_S + \alpha_D = 1$. Those weights can be user-defined so that to reflect the user's preference between interestingness and similarity. Also, notice that all objectives are normalized in the range $[0,1]$. Accordingly, the overall multi-objective utility function takes value in the same range (i.e.,$[0,1]$), where the goal is to maximize that overall utility under specified constraints. Such goal is formulated as follows:

Definition: Query Refinement for View Recommendation: *G*iven an user-specified query Q on a database D_B, a multi-objective utility function U, a significance level α, statistical power $1-\beta$ and a positive integer k. Find k aggregate views that have the highest utility values, from all of the refined queries $Q_j \in \mathbb{Q}$ such that $pvalue(V_{i,Q_j}) \leq \alpha$ and $power(Q_j) > 1 - \beta$.

In short, we premise that a view is of high utility for the user, if it satisfy the constraints, shows high deviation and is based on a query that is similar to the user's input query.

To estimate the cost incurred in solving the defined problem, note that for each refined query $Q_j \in \mathbb{Q}$, number of aggregate queries posed to the database equal to the number of aggregate views generated i.e., $2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. Furthermore, as discussed in previous section the number of refined queries is exponential to the number of predicates p. Therefore, with query refinement the total number of candidate views N are: $N \approx n^p \times 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$.

This is a very large search space and require an effective navigation by maintaining minimum cost. In Section 4.5 we propose different schemes to efficiently navigate the large search space for recommending top-k views.

4.4.3 Similarity Aware Query Refinement

In this section we define our metric for measuring similarity between input query and refined query. Eq 4.2 quantifies the change in the input query Q, by the refinement process, to get to a refined query Q_j, in terms of distance between Q and Q_j. In literature, number of methods have been proposed to measure the distance between two range queries. Specifically, distance between Q and Q_j can be measured in terms of Jaccard distance or edit distance between the predicates of the two queries [88]. However, it is very coarse and fails to differentiate between the change in the value of predicates. Another method is to measure the distance in terms of number of tuples changed in the two queries [92, 93]. This quantifies the exact change in underlying data of the queries, however, it is an expensive method because it requires database probes. Similar to [84, 85], we calculate the distance in terms of absolute change in predicate values. This method provides a reasonable approximation of the data at a negligible cost. Additionally, we normalize it by predicate bounds to take care of the different scales of various predicates.

$$s(Q, Q_j) = \frac{1}{p} \sum_{i=1}^{p} \frac{|l_i^{Q_j} - l_i^{Q}| + |u_i^{Q_j} - u_i^{Q}|}{2|U_i - L_i|} \tag{4.5}$$

Example: Consider Q and Q_1 of Example 2, the range of predicate education is [1,16], distance between the two queries is: $s(Q, Q_1) = \frac{|1-12|+|1-16|}{2|16-1|} = 0.86$. In other words this mean there is 14% similarity between Q and Q_1. While, Q_2 and Q has 87% similarity.

4.5 Search Schemes

For an input query Q, with a set of numeric dimension attributes $\mathbb{P}$, each possible query refinement of Q can be represented as a point in p-dimensional space, where $|\mathbb{P}| = p$ (please see Section 4.3.2 for more details). Clearly, one of the points in that space is the input query Q itself, and the remaining points belong to the set of refined queries $\mathbb{Q}$. Our high-level goal is to: 1) generate the set $\mathbb{Q}$, 2) compute the utility of all the aggregate views generated from each query in $\mathbb{Q}$, and 3) recommend the

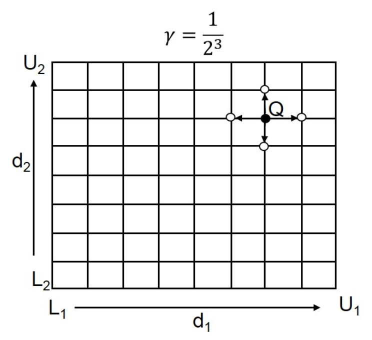


Figure 4.3: Query Space $\mathbb{Q}$

top-k views after ranking them based on their achieved utility. To that end, clearly the large size of $\mathbb{Q}$ and the corresponding aggregate views, together with the complexity of evaluating the statistical significance and utility function of each view, makes the problem highly challenging.

Hence, in this section, we put forward various search strategies for finding the top-k views for recommendation. We present the baseline scheme namely Linear (Exhaustive) together with our proposed schemes: i) Query Refinement for View Recommendation (QuRVe), ii)QuRVe with tighter upper bounds (uQuRVe), iii) QuRVe with pseudo sorted cardinality (uQuRVe-range) and iv) Approximation based schemes (uQuRVe(App)), (uQuRVe-range(App)).

4.5.1 Linear Scheme

Clearly, a naive way to identify the top-k objects is to score all those objects based on a scoring function, sort according to their scores, and return the top-k objects. Accordingly, the Linear scheme is basically an exhaustive, naive and brute force strategy, in which views from all possible queries are generated and ranked according to their utility.

As we consider predicates on continuous dimensions, infinite possible values can be assigned to predicates in refined queries. Therefore, each dimension is discretized with a user specified parameter γ. This divides the range of dimension attribute into $1/\gamma$ equi-width intervals. In this scheme, irrespective of Q, iteratively all possible refined queries are generated using all combinations of Predicates $P_1, P_2 ... P_p$. For instance, in Fig. 4.3, the grid of two dimensions is shown. All intersections points on the grid represent all refined queries at $\gamma = \frac{1}{2^3}$ having predicates on dimensions d_1 and d_2,.

Moreover, for each query $Q_j \in \mathbb{Q}$, to check the constraint $power(Q_j) < 1 - \beta$, a function *powerTest(*Q_j*,* ω*,* β*,* α*)* is defined. The function returns true value if the constraint is satisfied, otherwise it returns false. The cost of checking this constraint is one database probe, where a $COUNT$ query with predicates of Q_j is executed to get the sample size of Q_j.

Moreover, for the queries that satisfy the statistical power constraint, all views are generated. Then for each view V_{i,Q_j} the constraint $pValue(V_{i,Q_j}) \leq \alpha$ is checked. Specifically, for this purpose, another function *significanceTest(*V_{i,Q_j}*,* α*)* is defined, which calculates the p-value and returns a true value if p-value$\leq \alpha$. The view that satisfied the constraint, their utility value $U(V_{i,Q_j})$ is computed, and finally the top-k views are returned.

Algorithm 1 QuRVe.

Require: Input query Q, an integer k, stepSize γ, Significance level α, Power level β, Effect size ω, Similarity weight α_S.
Ensure: List of top-k views.

```
1: $U_{Seen} \leftarrow 0$
2: $U_{Unseen} \leftarrow 1$
3: $Q_{list} \leftarrow Q$
4: $Q_j \leftarrow Q_{list}.head$
5: while $!empty(Q_{list})$ & $U_{Unseen} > U_{Seen}$ do
6:     if $powerTest(Q_j, \omega, \beta, \alpha) == TRUE$ then
7:         $S_{list} \leftarrow generateViews(Q_j)$
8:         for each $V_{i,Q_j}$ do
9:             if $significanceTest(V_{i,Q_j}, \alpha) == TRUE$ then
10:                $computeUtility(V_{i,Q_j})$
11:                $update(topk)$
12:                $U_{Seen} \leftarrow k_{th}(topk)$
13:            else
14:                $reject(V_{i,Q_j})$
15:            end if
16:        end for
17:    else
18:        $reject(Q_j)$
19:    end if
20:    $Q_{list} \leftarrow refineQueries(Q_j, \gamma)$
21:    $Q_j \leftarrow Q_{list}.next$
22:    $U_{Unseen} \leftarrow \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u(V_i, Q_j)$
23: end whilereturn $topk$
```

4.5.2 The QuRVe Scheme

Clearly, the straightforward linear search scheme, described above, visits every possible view, therefore, it is very expensive in terms of execution time. In this section, we present the QuRVe scheme, which reduces cost by pruning a large number of views.

Notice that our problem of finding top-k views is similar to the problem of top-k query processing. Top-k query processing problem is an extensively studied area in various settings such as web accessible databases and multimedia similarity search [70]. Generally in these settings objects are evaluated by multiple objectives that contribute to the overall score of objects. In terms of efficiency, the best performing techniques for various top-k problem settings are based on the threshold algorithm (TA) [70, 71]. TA initially generates sorted lists of objects on partial scores for every objective. Then it visits the lists in round robin fashion and merges objects from different lists to compute the aggregated scores. Usually, it early terminates the search as soon as it has the $top-k$ objects, i.e., long before reaching the end of the lists.

In our settings, we have similar situation i.e., we have two partial scores of a view V_{i,Q_j}:,1) Similarity score $S(Q, Q_j)$, 2) Deviation score $D(V_{i,Q_j})$. These are stored in S_{list} and D_{list}. Conversely, we also have some key differences; Firstly, for any view V_{i,Q_j} the values of $S(Q, Q_j)$ and $D(V_{i,Q_j})$ are not physically stored and are computed on demand. Secondly, calculation of $D(V_{i,Q_j})$ for a view is an expensive operation. Thirdly, the size of the view search space is prohibitively large and potentially infinite.

Obviously, a forthright implementation of TA is infeasible to our problem due to the limitations mentioned before. However, recall that the similarity objective $S(Q,Q_j)$ is the comparison of predicates of Q_j with Q and involves no database probes. Hence, out of the two lists mentioned above, a sorted list S can be easily generated at a negligible cost.

However, populating D_{list} in a similar fashion is non-trivial, as it involves expensive database probes. Particularly, to minimize the number of probes and efficiently populate D_{list}, the Sorted-Random (SR) model of the TA algorithm [70] is employed. In SR model the sorted list (S) provides initial list of candidates and the random list (R) is probed only when required. Accordingly, QuRVe provides S_{list} as the initial list of candidate views by incrementally generating refined queries in decreasing order of similarity and populating the S_{list}. The views in S_{list} have their partial scores, consequently, the final scores are only calculated for the views for which the D_{list} is also accessed. To achieve this, QuRVe maintains the following two values:

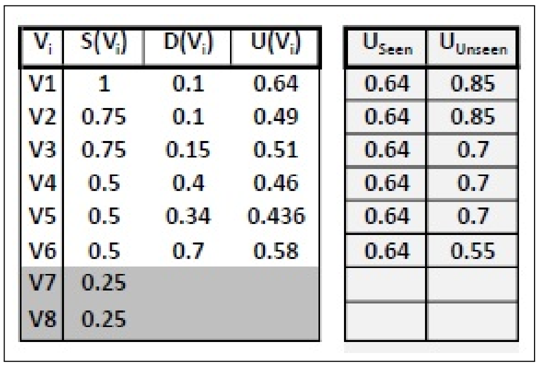
1. U_{Unseen}: Stores maximum utility of the views that are yet to be probed.

2. U_{Seen}: Stores the k^{th} highest utility of a view seen so far.

Specifically, for U_{Unseen}, consider V_{i,Q_j} as the next view in the similarity list and let the upper bound on its deviation be $D_u(V_{i,Q_j})$. The upper bound on deviation from all views be D_u then $D_u = Max[D_u(V_{i,Q_j})]$. Consequently, $U_{Unseen} = \alpha_S \times S(Q,Q_j) + (1-\alpha_S) \times D_u$. As the $D(V_{i,Q_j})$ is the normalized deviation, therefore theoretically $D_u = Max[D_u(V_{i,Q_j})] = 1$.

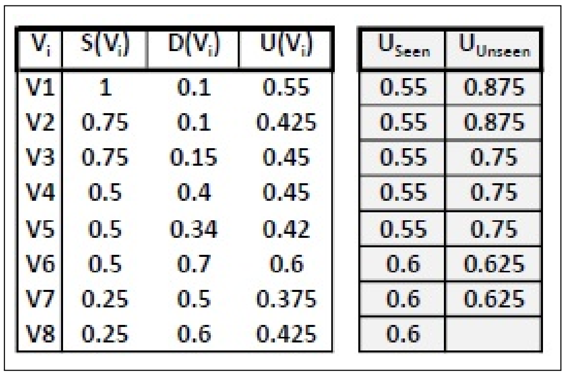
The main idea is to keep iterating following four steps. 1) Generate queries in decreasing order of similarity and add them to list of refined queries Q_{list}, 2) take a query from Q_{list} and add its corresponding views to the S_{list}, 3) take a view from S_{list} evaluate its utility, and 4) Update U_{Unseen} and U_{Seen}; Until U_{Unseen} becomes smaller than U_{Seen}, which means the remaining views have utility less than the utility of already seen views. Moreover, a list $topk$ is maintained which has the ordered top-k views seen so far.

In detail, QuRVe starts by making some initializations (Algorithm 1, line 1-4): (i) As there are no views generated yet, therefore maximum seen utility $U_{Seen} = 0$, (iii) The maximum any view can have is $U_{Unseen} = 1$, and (iii) The first query to be considered will be the input query Q as it has the highest similarity i.e., $S(Q,Q) = 1$. Therefore, Q is added to Q_{list} as the first member. Then, power of currently under consideration query Q_j is checked by calling the function *powerTest(Q_j, ω, β, α)*. Next, the corresponding views are generated by calling the function *generateViews(Q_j)* and the statistical significance test is performed on each view by calling the function *significanceTest(V_{i,Q_j}, α)*. Utility of the views that pass the test is computed. Accordingly the list $topk$ is updated. Additionally, the utility of k^{th} highest view is copied into U_{Seen}, to maintain the bound on the seen utility values. This completes processing the currently under consideration query Q_j.



V_i	$S(V_i)$	$D(V_i)$	$U(V_i)$	U_{Seen}	U_{Unseen}
V1	1	0.1	0.64	0.64	0.85
V2	0.75	0.1	0.49	0.64	0.85
V3	0.75	0.15	0.51	0.64	0.7
V4	0.5	0.4	0.46	0.64	0.7
V5	0.5	0.34	0.436	0.64	0.7
V6	0.5	0.7	0.58	0.64	0.55
V7	0.25				
V8	0.25				

(a) Example1



V_i	$S(V_i)$	$D(V_i)$	$U(V_i)$	U_{Seen}	U_{Unseen}
V1	1	0.1	0.55	0.55	0.875
V2	0.75	0.1	0.425	0.55	0.875
V3	0.75	0.15	0.45	0.55	0.75
V4	0.5	0.4	0.45	0.55	0.75
V5	0.5	0.34	0.42	0.55	0.75
V6	0.5	0.7	0.6	0.6	0.625
V7	0.25	0.5	0.375	0.6	0.625
V8	0.25	0.6	0.425	0.6	

(b) Example2

Figure 4.4: The QuRVe Scheme

Subsequently, the next set of neighboring queries in terms of parameter γ are generated. Particularly, the next query is generated by replacing each predicate $P_i \leq x_i$ with two predicates $P_i \leq x_i \pm \gamma$. The same process is repeated for each new query.

In next iteration, another query Q_j is taken from the Q_{list} in order of the S value and accordingly the value of U_{Unseen} is updated. The iterations (Algorithm 1, line 6-21) continue, until either there are no more queries to process (i.e., $empty(Q_{list})$ is true), or the utility of remaining queries will be less than already seen utility (i.e., $U_{Unseen} > U_{Seen}$ is false). If QuRVe terminates because of the first condition (i.e., $empty(Q_{list})$ is true) that means its cost is the same as Linear search, as the optimization did not get a chance to step in. However, often QuRVe terminates because of the second condition (i.e., $U_{Unseen} > U_{Seen}$ is false) and achieves early termination.

Example: Consider an example where we have 8 views ($V_1 - V_8$), from various refined queries. The views are ordered according to precomputed similarity, as shown in Fig. 4.4a by the column $S(V_i)$. The parameters are set as $k = 1$, $\alpha_S = 0.6$ and $\alpha_D = 0.4$. Additionally, $D(V_i)$,and $U(V_i)$ columns correspond to the deviation and utility values of the view V_i, which are computed on demand. For instance, when view V_1 is probed, its deviation is 0.1. Then the utility is computed as: $0.6 \times 1 + 0.4 \times 0.1 = 0.64$. In linear search all of the 8 views are probed to compute their deviation and utility values. However, for QuRVe scheme after probing V_6, $U_{Unseen} = 0.6 \times 0.25 + 0.4 \times 1 = 0.55$ while $U_{Seen} = 0.64$, therefore, the condition $U_{Unseen} > U_{Seen}$ becomes false, hence the search is early terminated and two of the views get pruned.

QuRVe reduces the cost by pruning unnecessary views. However, it is efficient with particular settings of input parameters or when the refined query that have the view with the maximum utility is near the input query. For instance, consider Fig. 4.4b, which shows same example of Fig. 4.4a but with $\alpha_S = \alpha_D = 0.5$. In this particular example, as the deviation has more weight, therefore, the value of U_{Unseen} is bigger, while U_{Seen} is smaller, as compared to Fig. 4.4a and as a result QuRVe fails to prune any views.

QuRVe is restricted by high value of U_{Unseen}. Particularly, the most efficient performance is expected when U_{Unseen} decrease quickly during search and early termination can be triggered. In Section 4.5.3, we propose uQuRVe which is particularly designed with the intention to ensure a quick decrease in U_{Unseen}.

4.5.3 The uQuRVe Scheme

Recall that the fundamental idea underlying our proposed QuRVe scheme is to early terminate the search based on the upper bound on the utility of not yet seen views (i.e., U_{Unseen}). Particularly, U_{Unseen} is computed by using the similarity value of the next view on the list, together with the upper bound on deviation D_u. Hence, if U_{Unseen} is less than the already seen maximum utility (U_{Seen}), then all of the remaining views have no chance of coming up to the top-k. Therefore, QuRVe terminates the search at that point, and those remaining views are pruned. Note that the D_u used by QuRVe to calculate U_{Unseen} is basically the theoretical extreme value and it is oblivious to the analyzed data. Hence, in practice, that upper bound is typically loose as it tends to overestimate the upper bound on deviation between the target and comparison views generated from the analyzed data. Consequently, many chances of pruning unnecessary views are missed.

To improve the pruning power, we note that in our problem setting views are generated corresponding to the input query Q, as well as all its refinements (i.e., $Q_j \in \mathbb{Q}$). In that setting, we observe that while the target views for every Q_j are generated from scratch (i.e., require a separate query execution), the comparison views are only generated once for the input query Q and those same comparison views are reused later for each Q_j. Such observation is the main idea underlying our novel uQuRVe scheme introduced in this section. Particularly, uQuRVe leverages such observation to provide a tighter upper bound on deviation by using those already executed comparison views, together with the properties of the deviation function, as explained next.

We first outline the properties of our deviation function to provide a tighter bound on deviation. Specifically, according to Eq 4.1 any distance metric can be used for computing deviation, we take Euclidean distance as our metric. Let V_c and V_t be our comparison and target views respectively with c categories, then the squared Euclidean distance is:

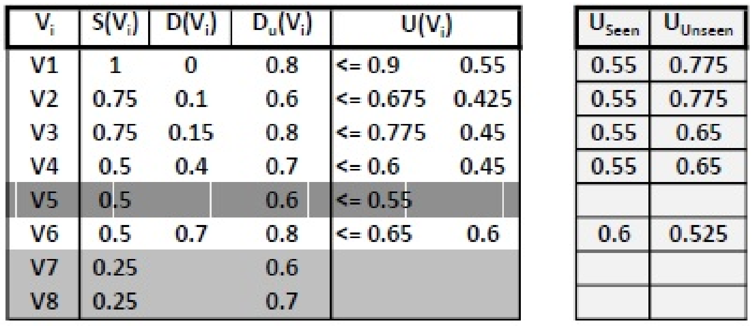
$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} (P_{V_c}[x] - P_{V_t}[x])^2$$

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} P_{V_c}[x]^2 + \sum_{x=1}^{c} P_{V_t}[x]^2 - 2 \times \sum_{x=1}^{c} (P_{V_c}[x] \times P_{V_t}[x]) \tag{4.6}$$

Recall, in the previous scheme QuRVe, D_u the upper bound on deviation is the theoretical maximum value of the distance between V_t and V_c. This maximum value is achieved when the last term in Eq. 4.6 is zero and consequently, $dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} P_{V_c}[x]^2 + \sum_{x=1}^{c} P_{V_t}[x]^2$. To be precise this maximum value is only possible when for each category x either $P_{V_c}[x]$ or $P_{V_t}[x]$ is zero. Resultantly, $dist^2_{P_{V_c},P_{V_t}} = 2$ and $D_u = \sqrt{2}$.

However, this is the theoretical maximum, when the exact probability distribution of V_c and V_t are unknown. While, in our problem settings, the P_{V_c} is known from the already executed comparison views for Q. Hence, we propose uQuRVe scheme, that takes advantage of already calculated P_{V_c} and calculate more realistic individual upper bounds $D_u(V_{i,Q_j})$ and overall upper bound D_u.

Particularly, let the upper bound on deviation corresponding to a comparison view be $D_u[V_c]$. The main idea is to calculate the $D_u[V_c]$ for each comparison view and use it later for two purposes: 1)



V_i	$S(V_i)$	$D(V_i)$	$D_u(V_i)$	$U(V_i)$	
V1	1	0	0.8	<= 0.9	0.55
V2	0.75	0.1	0.6	<= 0.675	0.425
V3	0.75	0.15	0.8	<= 0.775	0.45
V4	0.5	0.4	0.7	<= 0.6	0.45
V5	0.5		0.6	<= 0.55	
V6	0.5	0.7	0.8	<= 0.65	0.6
V7	0.25		0.6		
V8	0.25		0.7		

U_{Seen}	U_{Unseen}
0.55	0.775
0.55	0.775
0.55	0.65
0.55	0.65
0.6	0.525

Figure 4.5: uQuRVe Example

calculate upper bound on the utility of a target view, which can result in short circuiting of that view, and 2) calculate the value of U_{Unseen}, which can result in early termination of the search.

Note that the query for the comparison view has been executed and its $P[V_c]$ is known. h However, without executing the target query, we assume a hypothetical target probability distribution $P[V_t]$ such that it will result in the maximum value of deviation. Particularly, the upper bound $D_u[V_c]$ will be achieved when for the one category in P_{V_c} having the minimum value, the corresponding value in P_{V_t} is maximum (i.e., 1.0). For this to happen, and since the overall $\sum_{x=1}^{c} P_{V_t}[x]$ has to be equal to 1.0, then all other values in P_{V_t} have to be 0.0.

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} (P_{V_c}[x])^2 + \sum_{x=1}^{c} (P_{V_t}[x])^2 - 2 \times Min(P_{V_c}[x] \times 1) \tag{4.7}$$

For instance. assume a comparison view having four categories and $P_{V_c} = [0.3, 0.4, 0.1, 0.2]$. The minimum value is in the third category therefore to have a maximum deviation let the hypothetical target view have $P[V_t] = [0, 0, 1, 0]$. Resultantly, the squared Euclidean distance will be:
$dist^2 = (0.3^2 + 0.4^2 + 0.1^2 + 0.2^2) + (1^2) - (2 \times 0.1 \times 1) = 1.1$

The normalized upper bound on deviation $D_u[V_c]$ will be 0.7. As the theoretical upper bound is $\sqrt{2} = 1.4$, hence, our new upper bound is 50% less than the theoretical upper bound.

Once the $D_u[V_c]$ is calculated for all comparison views, D_u is assigned the maximum value from all of the comparison views i.e., $D_u = Max(D_u[V_c])$. The D_u remains fixed for all iterations as the views are accessed in order of their similarity value and there is no order on the deviation or upper bound of deviation of the views. Then U_{Unseen} is updated accordingly as $U_{Unseen} = \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u$. The $D_u(V_{i,Q_j})$ is used to calculate the upper bound on utility of V_{i,Q_j}. Particularly, before executing the target query, the upper bound on utility of a view is calculated as $U(V_{i,Q_j}) \leq \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u(V_{i,Q_j})$. If $U(V_{i,Q_j}) \leq U_{Seen}$ then the target query is not executed and that view is pruned (short circuit). uQuRVe performs better than QuRVe firstly by achieving earlier early termination and secondly by pruning of many unnecessary views through short circuits.

Example: Consider the example of Fig. 4.4b again. Now we have calculated the upper bound on deviation for all views as shown by column $D_U(V_i)$ in Fig. 4.5, and $D_u = Max(D_U(V_i)) = 0.8$. Similar to QuRVe at the start $U_{Seen} = 0$ and $U_{Unseen} = 1$. The first view in S_{list} is V_1, initially it is checked for short circuit. Particularly, its upper bound of utility is computed as $U <= 0.5 + 0.5 \times 0.8 = 0.9$. As the upper bound is greater than U_{Seen}, therefore short circuit is not possible, it is probed for deviation

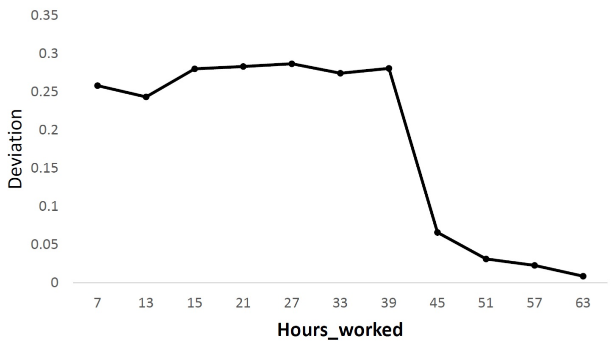


Figure 4.6: Pattern of Deviation

and actual value of U is computed. After probing V_1 to V_4 in order of similarity, $U_{Seen} = 0.55$ and $U_{Unseen} = 0.65$. For V_5 when the upper bound on utility is computed using $D_u(V_5)$, it is less than U_{Seen}. Therefore, V_5 is short circuited and it's deviation is not probed. Afterwards, after probing V_6, U_{Unseen} becomes smaller than U_{Seen}, therefore the search is early terminated. In comparison to QuRVe which probed 8 views, uQuRVe probed only 5 views.

The QuRVe and uQuRVe schemes have S_{list} which provides sorted(S) access only and D_{list} which provides random(R) access only. The D_{list} is accessed when the view under consideration can have a utility greater than U_{Seen}. Specifically, the access to D_{list} means calculating the statistical significance and eventually the deviation value for the view by generating the database probes for target query. The search can be further optimized by having higher U_{Seen} value earlier in search.

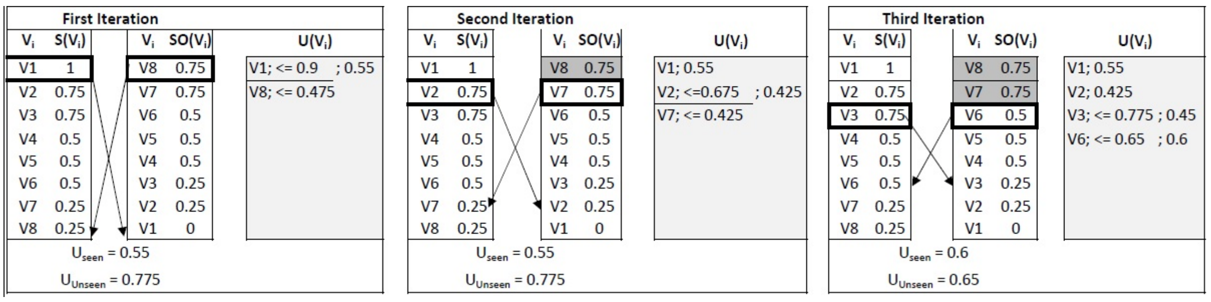
4.5.4 The uQuRVe-range Scheme

QuRVe and uQuRVe schemes were based on the Sorted-Random (SR) model of the Threshold algorithm (TA), as having D_{list} sorted is very expensive and contrary to our objective. However, as mentioned in Section 4.5.2, the best performing techniques for top-k problem settings are based on the threshold algorithm (TA) [70, 71].

The main idea behind this scheme is to reduces even further by having more control on the ordering of probed views and first probe the views with the high utility. As the utility of a view depends on the similarity and deviation objective values. The views are already accessed in the decreasing order of similarity, however previous schemes lack effort to ensure any order on the deviation. Hence, we studied the pattern of deviation for different dimensions. Although, deviation of a view does not have a strict pattern but it shows decreasing trend as we increase range of the refined queries, which means as the cardinality of the queries increases, the deviation generally decreases. For example, Fig. 4.6 shows pattern of deviation values for a single dimension attribute `hours_worked`. Accordingly, we define a metric $S_O(V_i)$ in terms of similarity of the underlying query of view V_i with the most restrictive refined query. Let the most restrictive query having minimum cardinality be Q_O having all predicates as $l_i \leq P_i \leq l_i$.

$S_O(V_i) = 1 - DistFromQ_O(V_i)$.

In the uQuRVe-range scheme a SO_{list} which records similarity with Q_O is also maintained along with the S_{list} and D_{list}. The S_{list} provides sorted and random(SR) access, SO_{list} provides sorted (S)



First Iteration

V_i	$S(V_i)$	V_i	$SO(V_i)$	$U(V_i)$
V1	1	V8	0.75	V1; <= 0.9 ; 0.55
V2	0.75	V7	0.75	V8; <= 0.475
V3	0.75	V6	0.5	
V4	0.5	V5	0.5	
V5	0.5	V4	0.5	
V6	0.5	V3	0.25	
V7	0.25	V2	0.25	
V8	0.25	V1	0	

$U_{seen} = 0.55$

$U_{Unseen} = 0.775$

Second Iteration

V_i	$S(V_i)$	V_i	$SO(V_i)$	$U(V_i)$
V1	1	V8	0.75	V1; 0.55
V2	0.75	V7	0.75	V2; <=0.675 ; 0.425
V3	0.75	V6	0.5	V7; <= 0.425
V4	0.5	V5	0.5	
V5	0.5	V4	0.5	
V6	0.5	V3	0.25	
V7	0.25	V2	0.25	
V8	0.25	V1	0	

$U_{seen} = 0.55$

$U_{Unseen} = 0.775$

Third Iteration

V_i	$S(V_i)$	V_i	$SO(V_i)$	$U(V_i)$
V1	1	V8	0.75	V1; 0.55
V2	0.75	V7	0.75	V2; 0.425
V3	0.75	V6	0.5	V3; <= 0.775 ; 0.45
V4	0.5	V5	0.5	V6; <= 0.65 ; 0.6
V5	0.5	V4	0.5	
V6	0.5	V3	0.25	
V7	0.25	V2	0.25	
V8	0.25	V1	0	

$U_{seen} = 0.6$

$U_{Unseen} = 0.65$

Figure 4.7: uQuRVe-range Example

access and D_{list} provides random(R) access only. S_{list} and SO_{list} are accessed in round robin fashion and D_{list} is only accessed when required. Similar to other schemes, U_{Unseen} and U_{Seen} are maintained and when $U_{Unseen} > U_{Seen}$ is false the search is terminated.

Example: Consider the same example of Fig. 4.4b, assume the views $V_1 - V_8$ correspond to the queries $Q_1 - Q_8$ respectively. Also assume Q_9 is Q_O and the SO_{list} has similarity values according to that. Now we have S_{list} sorted on the similarity value and a SO_{list} sorted on the the similarity with Q_O as shown in Fig.4.7. Similar to previous schemes at the start $U_{Seen} = 0$ and $U_{Unseen} = 1$. uQuRVe-range access the lists in round robin fashion. Therefore, in first iteration in Fig.4.7, first view in S_{list} is V_1,. After a failed short circuit test its utility is computed and U_{Seen} is updated. Then it goes to the SO_{list} which has V_8 as the first member. V_8 is pruned as its upper bound on utility is less than U_{Seen}. In second iteration V_7 is pruned and in third iteration the algorithm early terminates. uQuRVe-range probes only 4 views as compared to 5 of uQuRVe.

uQuRVe-range reduces the cost even more than uQuRVe because it first visits the views which have high probability of having top-k utilities. Consequently, it sets U_{Seen} to a high value and the condition $U_{Unseen} > U_{Seen}$ becomes false very early as compared to uQuRVe. uQuRVe-range Scheme becomes even more useful in a special case when initial hypothesis i.e., input query is unavailable. This means that the similarity metric is irrelevant in this case and α_S should be 0 . Consequently, the utility value completely depends on the deviation objective. In QuRVe scheme there the D_{list} access is random therefore it will actually perform same as the linear scheme. However, uQuRVe-range has the advantage of having SO_{list} which will partially ensure that views are accessed in order of high utility.

4.5.5 The QuRVe-Approximation

All the search algorithms presented so far are accurate as they provide the same top-k views as the baseline linear search. In the following, we introduce an approximation extension for uQuRVe and uQuRVe-range schemes to further improve performance, while incurring negligible loss in the quality of recommendation.

As explained earlier the pruning power of a scheme depends heavily on the theoretical upper bound. In reality the actual upper bound is much smaller than the one used in these schemes. However, it is not practical to calculate the actual upper bound, as it requires probing all of the views which conflicts with the goal of reducing cost through pruning. In this approximation approach we propose to estimate the D_u with high accuracy. The distribution of deviation is generally skewed and it does not follow a normal distribution. Therefore, we consider non-parametric predictive interval model to determine the D_u with certain level of confidence without any assumption on the population. A prediction interval is an estimate of an interval in which a future observation will fall, with a certain probability, given what has already been observed. We execute sample views and the maximum deviation calculated in those views is taken as the upper bound and assigned to D_u. If there are y samples, then $\frac{y-1}{y+1}$ is the prediction interval. We determine the number of samples y from the predictive interval. For instance, setting $y = 20$ results in a 90% predictive interval. Which means 90% of the time the actual deviation will be less than the upper bound set on deviation through samples. Obviously the higher the number of sample is, higher the accuracy of D_u. However, this will also result in increase in cost.

Applying the approximation approach to uQuRVe, the first y views are executed from the S_{list} and after that D_u is set accordingly. While for uQuRVe-range scheme when this approximation is applied, y views are executed from S_{list} and SO_{list} in a round robin fashion, then the D_u is updated. The D_u is later updated if a view is encountered which has deviation greater than the current value of D_u. The approximation is particularly exceptional for uQuRVe-range scheme, when Q is far from Q_j having optimal view. In that case uQuRVe-range takes advantage of the ordering in the SO_{list} and reduces cost without the loss of accuracy.

4.6 Experimental Testbed

Parameter	Range	Default
SimilarityWeight (α_S)	0.0-1.0	0.5
top-k (k)	1-25	10
Grid resolution (γ)	$1/2^4$-$1/2$	$1/2^3$
Number of predicates (p)	1-4	3

We perform extensive experimental evaluation to measure both the efficiency and effectiveness of the different search strategies of query refinement for top-k view recommendation. Here, we present the different parameters and settings used in our experiments.

Setup: We built a platform for refining query and recommending visOur experiments are performed on a Corei7 machine with 16GB of RAM memory. The platform is implemented in Java, and PostgreSQL is used as the backend database management system.

Data Analysis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. We use *CENSUS:* the census income dataset [20] and *FLIGHTS:* the flight delays dataset [94] . The CENSUS dataset has 14 attributes and 48,842 tuples. The independent categorical attributes of the dataset are used as dimensions (e.g., occupation, work class, hours per week, sex, etc.), whereas the observation attributes are used as measures (capital gain, capital loss, etc.) and the numerical independent attributes are used for predicates (e.g., education, age, etc.). The CENSUS dataset is used as default dataset for experiments. The FLIGHTS dataset has 18 attributes and more than 5M tuples.

In our default setting, $|\mathbb{A}| = 3$, $|\mathbb{M}| = 3$, $|\mathbb{F}| = 3$ and $p = 3$, where p is the number of predicates used in refinement. The aggregate functions used are SUM, AVG and COUNT. In the analysis, all the α_S is in the range $[0-1]$, where $\alpha_S + \alpha_D = 1$. In default settings $\alpha_S = 0.5$, $k = 10$ and $\gamma = \frac{1}{2^3}$. For the purpose of statistical significance in experiments we use chi-square goodness of fit test, which is the standard test for comparing difference between sample and population data for categorical dimension attribute.

Schemes: Firstly, as a baseline scheme we include SeeDB [6], in which no weight is given to similarity and the goal is maximize the sum of k deviations. Secondly, we use Hill Climbing (HC), with halving search as another baseline method [69]. We compare the performance of the baseline schemes (*Linear, SeeDB, HC*) with the schemes proposed in Section 4.5(*QuRVe, uQuRVe, uQuRVe-range, uQuRVe(App), uQuRVe-range(App)*).

Performance: We evaluate the efficiency and effectiveness of the different recommendations strategies in terms of cost incurred.

Cost: As mentioned in Section 4.4, the cost of a strategy is the total cost incurred in processing all the candidate views. We use the total views probed and execution time as the cost metric.

Overall Utility: It is the sum of the utilities of the top-k views.

Each experiment is performed with 10 randomly generated input queries, spread around the search space defined by predicates in $\mathbb{P}$, then average of the cost and overall utility is taken.

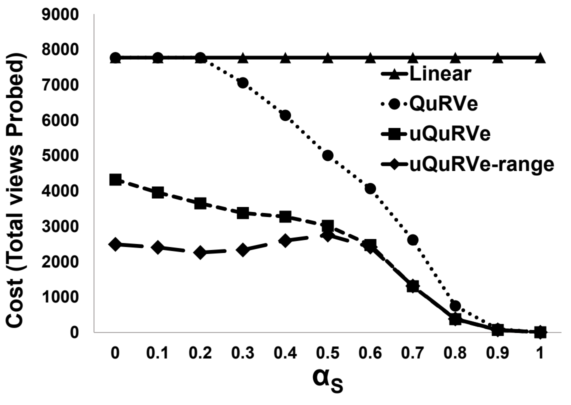
4.7 Experimental Evaluation

Impact of the α parameters (Fig. 4.8): In this set of experiments, we measure the impact of the α values on cost (i.e., total number of views probed and execution time). Figures 4.8 shows how the cost of *Linear*, *QuRVe*, *uQuRVe* and *uQuRVe-range* schemes are affected by changing the values of α_S for CENSUS and FLIGHTS datasets. In Fig. 4.8, α_S is increasing while α_D is implicitly decreasing. Notice that no approximations are employed in these schemes, hence, all schemes have the same overall utility.

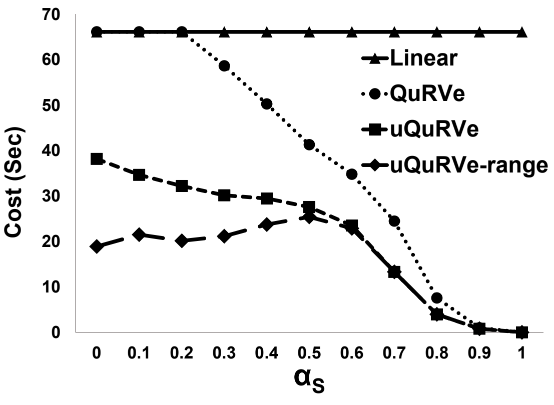
Fig. 4.8 shows that the Linear scheme has same cost for all values of α_S, which is as expected since it performs exhaustive search over all combinations of refined queries, dimensions, measures and aggregate functions. Therefore, its cost depends on the number of all combinations, irrespective of the value α.

Fig. 4.8a shows cost in terms of number of views probed, it shows that *QuRVe* has almost same cost as *Linear* for $\alpha_S = 0$ and $\alpha_S = 0.1$, but outperform it as the value of α_S increases. This happens because in the *QuRVe* scheme, the upper bound on deviation is set to the theoretical maximum bound i.e., $D_u = 1$ and when $\alpha_S = 0$, $U_{Unseen} = 0 \times S + 1 \times D_u = 1$, consequently early termination is not possible. On the contrary, as α_S increases, chances of applying the early termination condition based on the similarity value becomes possible. Consequently, this prunes many of the database probes.

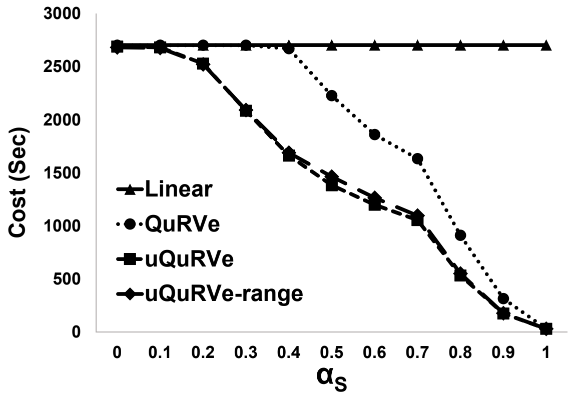
The amount of achieved pruning is further increased for $\alpha_S \leq 0.8$ under *uQuRVe*, because of its tighter upper bound on deviation. Consequently, this results in earlier early termination and number of short circuits on deviation calculation. For instance, in Fig. 4.8a at $\alpha_S = 0.3$, *uQuRVe* shows around



(a) CENSUS: Views Probed



(b) CENSUS: Execution Time



(c) FLIGHTS: Execution Time

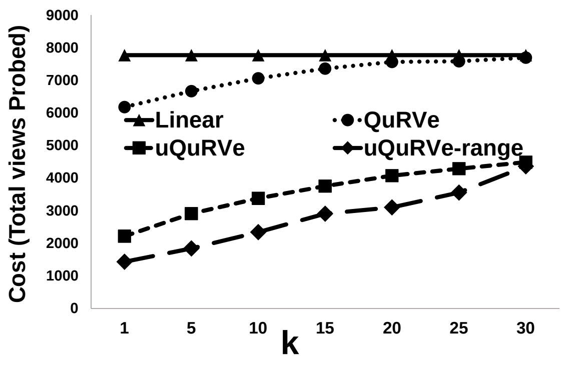
Figure 4.8: Impact of α_S and α_D on Cost

50% reduction in cost as compared to the *QuRVe*. The cost is further reduced in *uQuRVe-range*, which is able to prune deviation calculations even more because it visits the most probable top-k views first before low utility views. For instance, Fig. 4.8a shows that *uQuRVe-range* reduces the processing cost by more than 35%, compared to *uQuRVe*, at $\alpha_S = 0.2$. Notice *uQuRVe* and *uQuRVe-range* are able to prune views for all values of α due to the tighter upper bound on utility.

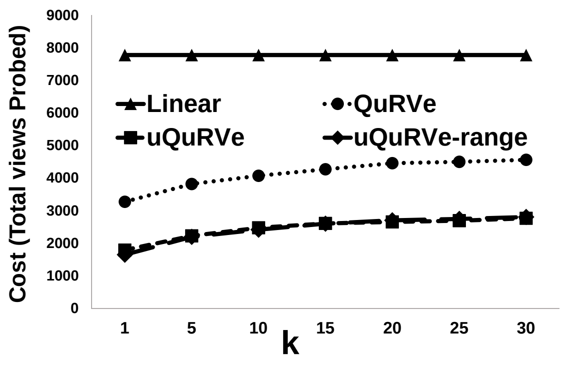
Fig. 4.8b shows the cost for CENSUS dataset in terms of execution time. The figure clearly shows that the pattern for all schemes is the same as Fig. 4.8a, this is because the execution time is mainly the I/O time and it is directly proportional to the number of probed views.

Fig. 4.8c shows the execution time for FLIGHTS dataset, it shows that the execution time for *Linear* is very high as compared to Fig. 4.8b because of the size of dataset. Moreover, the *uQuRVe* has lower execution time compared to the *QuRVe* scheme for all values of α_S due to the optimized upper bounds. However, for $\alpha_S \leq 0.5$ *QuRVe* is same as linear because the dataset lacks high deviation views, the maximum deviation found is 0.25, which results in low value of U_{Seen} and consequently missed early termination opportunities. However, the cost of *uQuRVe* and *uQuRVe-range* is almost the same because for this dataset the low cardinality views which are expected to have high deviation failed the statistical tests and as a result the optimization of *uQuRVe-range* scheme did not get the opportunity to prune views.

Impact of k (Fig. 4.9): Fig. 4.9a & 4.9b shows that *Linear* scheme is insensitive to the increase in the value of k. This is because it visits all views and sort them according to their utility irrespective of the

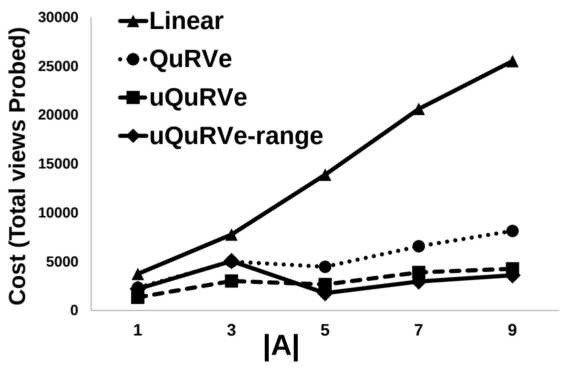


(a) $\alpha_S = 0.3$

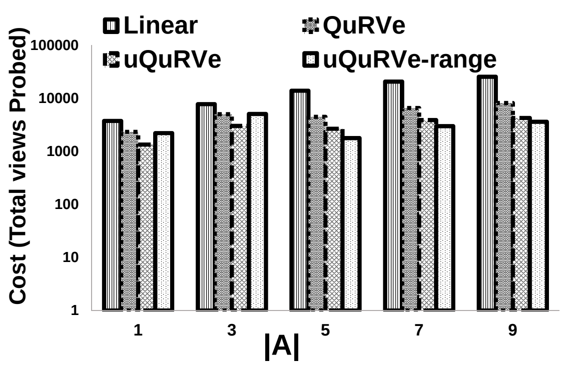


(b) $\alpha_S = 0.6$

Figure 4.9: Impact of k



(a) Normal Scale



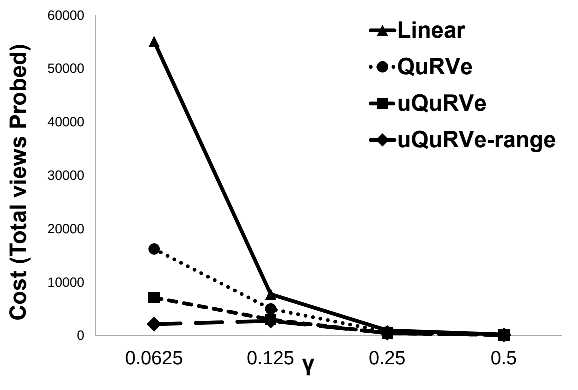
(b) Log Scale

Figure 4.10: Impact of Dimensions on Cost

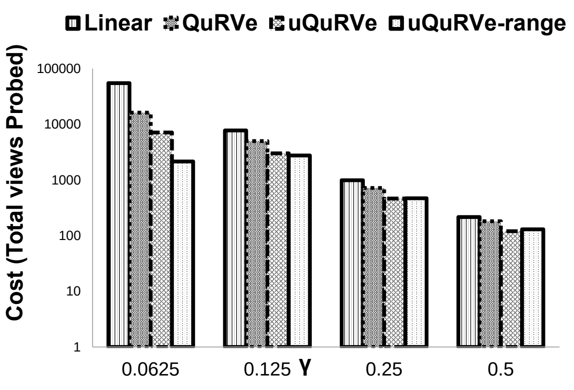
value of k.

For Fig. 4.9a, $\alpha_S = 0.3$, *QuRVe* performs better than *Linear* for small values of k, however as k increase it almost performs same as *Linear*, because *QuRVe* uses maximum upper bounds on deviation. Fig. 4.9a also shows that *uQuRVe* has lower cost than *QuRVe* for all values of k. This is because as soon as *uQuRVe* has seen the top-k highest utility views, early termination will be enabled leading to pruning many unnecessary low utility views. For instance, in case of top-1 *uQuRVe* reduces cost by up to 65% compared to the *Linear* scheme. Moreover, *uQuRVe-range* performs even better than *uQuRVe* for all values of k because of its psudo ordering on the deviation list. In Fig. 4.9b, $\alpha_S = 0.6$, all *QuRVe* schemes performs many folds better than *Linear*, and the performs remains stable even with the increasing number of k. This is because for $\alpha_S > 0.5$, the early termination always triggers for QuRVe schemes and prune many unnecessary views.

Scalability (Fig. 4.10- 4.12): The search space of our problem depends on p, γ, $|A|$, $|M|$ and $|F|$. Increasing any of these factors explodes the search space. Consequently, cost of all schemes increases because there are more views that are visited in search for top-k views. Fig. 4.10 shows the impact of number of dimensions on cost, particularly for this experiment number of dimensions ($|A|$) are increased from 1 to 9. Fig. 4.10 shows that the increase in the cost of *Linear* is linear with the increase in $|A|$. However, for *QuRVe* and *uQuRVe* the increase in the cost is slow. Particularly, the cost of *QuRVe*



(a) Normal Scale



(b) Log Scale

Figure 4.11: Impact of Grid Resolution on Cost (Log Scale)

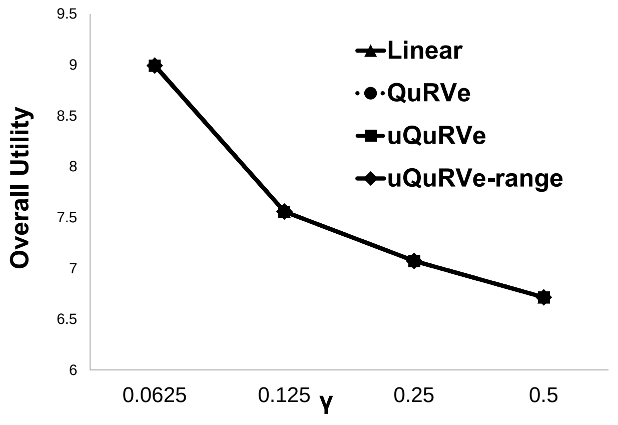
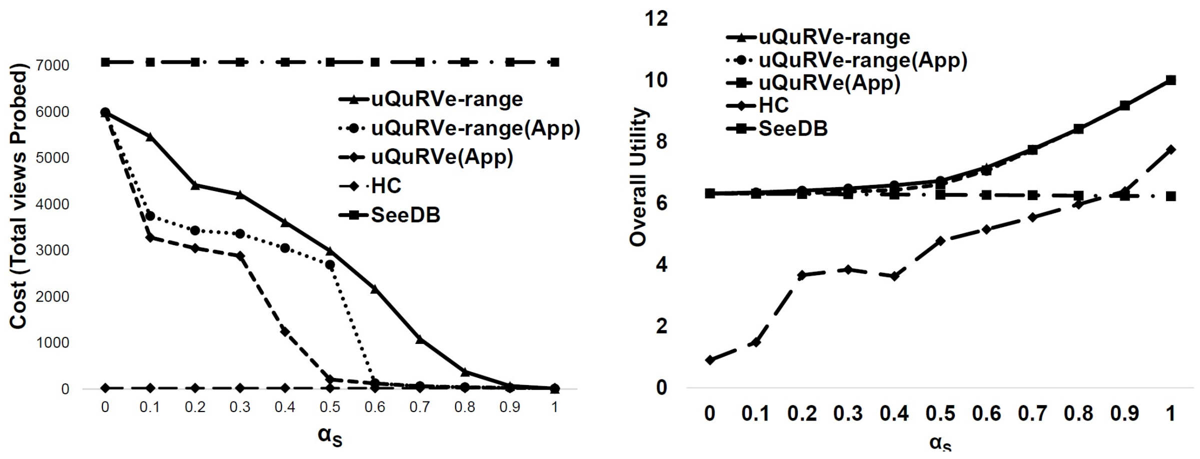


Figure 4.12: Impact of Grid Resolution on Overall Utility

and *uQuRVe* increases as the number of dimension are increased from 1 to 3, because the schemes search from more views to generate top-k views. But for $|A| = 5$ the cost of both schemes decreases this is because a new dimension is added which had views with high deviation and that resulted in increasing the value of *Seen* and triggering early termination. The cost of *uQuRVe-range* scheme is higher than that of *uQuRVe* for $|A| = 1-3$, the reason behind this is *uQuRVe-range* experiences an overhead of probing extra views as compared to *uQuRVe* when it takes turn between s-list and d-list. This overhead is mostly dominated by the savings of *uQuRVe-range* scheme. However, in the case under discussion the pseudo ordering on d-list failed to ensure a higher value of *Seen* early in search and that added overhead in the cost.

As mentioned in Section 4.5, we consider discretize the continuous dimensions by the user specified parameter γ. Fig. 4.11- 4.12 show the impact of this discretization factor or grid resolution on cost and overall utility value. A large value of γ represents sparse grid with bigger width cells, which means there are few refined queries. As γ decreases the cell width in grid decreases, which means the number of refined queries increases. More refined queries generally imply increase in cost but it should also be able to improve the overall utility of the top-k views.

Fig. 4.11 clearly shows that as the grid becomes dense the cost of *Linear* scheme increase because it has to search more number of refined queries for top-k views. The reduction in cost by *QuRVe*



(a) Cost (b) Overall Utility

Figure 4.13: Impact of Approximation Schemes

schemes gets higher with the decrease in γ. This is because the number of possible refined queries increases and it gives *QuRVe* schemes more opportunities to prune unnecessary views. Fig. 4.11b shows the same results of Fig. 4.11a by using a log scale on the y-axis. In Fig. 4.11b the performance gains of *QuRVe* schemes become even more clearer. Fig. 4.12 shows overall utility i.e., the sum of utiltiy of top-k views. All three schemes have the same overall utility as all schemes recommend the same top-k views, It can be clearly seen that as γ decreases the overall utility increases, which implies the denser grid resolution ensures increase in overall utility,

Approximations Techniques ((Fig. 4.13): As mentioned in Section 4.6, we consider SeeDB and Hill climbing (HC) for baseline comparison. These schemes do not give the optimized overall utility according to our defined utility metric, therefore we include them in approximation schemes. We show our best optimized scheme *uQuRVe-range* for comparison with approximation schemes *uQuRVe(App)* and *uQuRVe-range(App)*. Fig. 4.13a& 4.13b shows the cost and overall utility of different schemes. Fig. 4.13a shows that *SeeDB* has the same and maximum cost for all values of α, because it is based on purely deviation metric and no weight is given to similarity. Therefore, its cost depends on the number of all possible combinations, irrespective of the values of α. Fig. 4.13a also shows that the *HC* scheme provides the same cost regardless of α value. This is because *HC* is a local search based scheme, which is expected to hit a local maxima, hence, always has low cost. Meanwhile, the cost of *uQuRVe(App)* and *uQuRVe-range(App)* is lower than *uQuRVe-range* due to the employees approximation. For instance, at $\alpha_S = 0.1$, *uQuRVe(App)*scheme prunes almost 50% more views than *uQuRVe* scheme. Fig. 4.13b shows SeeDB start to fall back on overall utility for $\alpha \geq 0.5$. This is because SeeDB is purely based on deviation metric, when the α_D starts decreasing, the views selected by SeeDB lacks the similarity value, hence, overall utility start to decrease as compared to the optimal overall utility of *uQuRVe-range*. For *HC*, overall utility improves when $alpha_S$ is high. This is simply because *HC* search is guided by the halving intervals of similarity list instead of the utility values. As mentioned earlier *HC* is a local search based scheme, therefore, it fails to achieve the global maxima, which is reflect in its overall utility. *uQuRVe(App)* and *uQuRVe-range(App)* has overall utility almost the same as *uQuRVe-range* scheme.

4.8 Summary

Motivated by the need for visualizations recommendation that lead to interesting discoveries and avoid common pitfalls such as random or false discoveries, In this paper we formulated the problem of query refinement for view recommendation and proposed QuRVe schemes for view recommendation. QuRVe refines the original query in search for more interesting views. It efficiently navigates the refined queries search space to maximize utility and reduce the overall cost. The cost is further reduced with uQuRVe scheme, which applies tight upper bounds to prune more views. We also proposed uQuRVE-range scheme that navigates the search space close to the order of highest utility views. Additionally, we presented an approximation technique that further increase the cost savings provided by uQuRVe and uQuRVe-range. Our experimental results show that employing the QuRVe schemes offer significant reduction in terms of cost.

Chapter 5

View-360: A Prototype System for View Recommendation

5.1 Introduction

Visual data exploration is the most valuable and widely used tool in the analyst's toolbox. However, with growing size and complexity of data, manual visual data exploration becomes challenging. Therefore, to address the issue of efficiently and effectively identifying interesting visualizations, in this thesis, we have proposed visualization recommendation schemes targeting various aspects of the problem. Particularly, in this chapter, first the most relaxed version of the view recommendation problem as defined in Definition 4 is formulated. Specifically, we argue that in existing recommendation systems, the assumption that a comparison view is generated from a given reference dataset, limits the discovery of interesting insights. Therefore, the reference dataset (i.e., the query that selects the reference dataset) for generating the comparison views should also be refined. Second, the design and implement of a holistic prototype system *View-360* for view recommendation is presented. Third, effectiveness of all of our proposed schemes in this thesis is presented by performing analysis on real datasets from various domains.

Recall, that the baseline problem of view recommendation as stated in Definition 1, is that an analyst chooses a subset of data for visual analysis by providing an input Query Q. Specifically, for recommending top-k aggregate views, target and comparison views on all combinations of dimensions (A), measures (M), and aggregate functions (F) are generated, their utility is evaluated according to deviation-based utility function and top-k views are recommended. Specifically, for an aggregate view V_i, the target view $V_i(D_Q)$ is generated on the subset of data specified by Q, while the comparison views $V_i(D_B)$ is generated on the reference dataset which is typically the complete database.

The first dimension of the problem explored in Chapter3, is defined by Definition 2, where we identify and address the challenges of having numerical dimensions in the exploration. We formalized the view recommendation problem in the presence of numerical dimensions. Our proposed search schemes automatically binned the numerical dimensions and efficiently recommended the top-k views

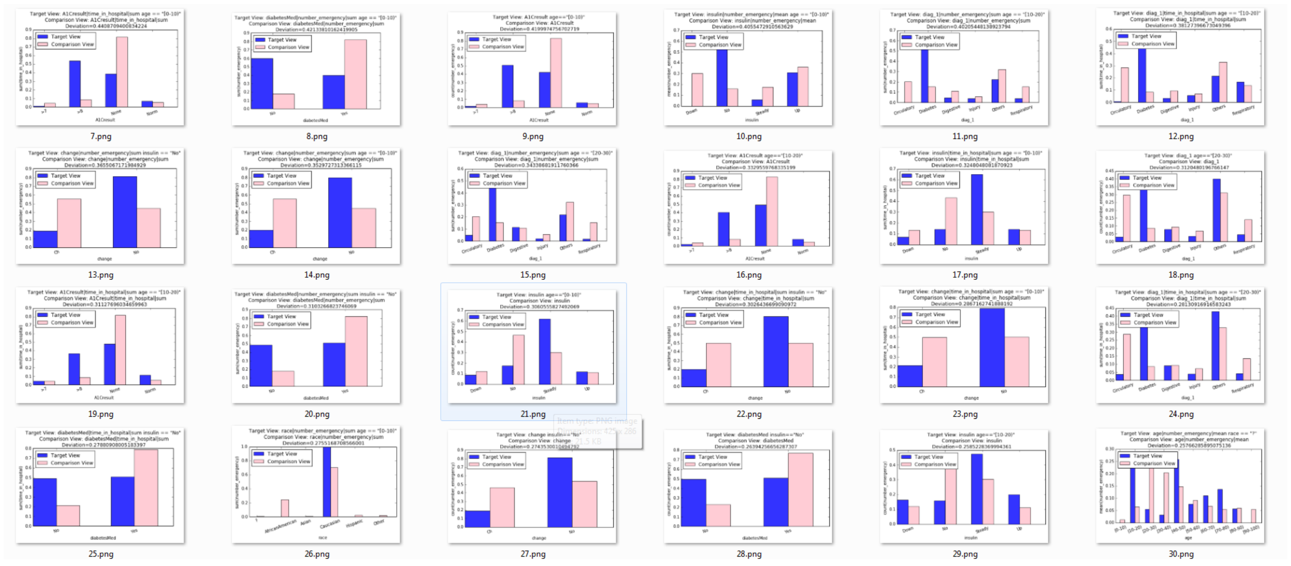


Figure 5.1: View-360: Recommended Views

including binned aggregate views. Moreover, During our analysis with various datasets, we realized that the condition of providing an input query is contradictory to the automatic view recommendation objective, as it restricts the search to the subset selected by Q. Additionally, mostly the analyst is only able to provide an uncertain input query, while the interesting views might belong to some other subset. Taking this observation further, in Chapter 4, we proposed efficient search schemes that automatically refine the input query. This ensured that the target views of the top-k aggregate views can come from any subset of the data.

As mentioned before, the top-k aggregate views are recommended, after computing deviation between target views defined on refined queries and comparison views defined on reference dataset. However, this fails to consider the aggregate views that can be generated by having both target and comparison views on the different subsets of data. Therefor, to explore this dimension of the problem in this chapter, we relax the assumption that the comparison views are generated from the predefined reference dataset. Instead the query used for comparison view is also automatically refined. There are two challenges associated with the refinement of reference dataset: Firstly, it increases the search space of views exponentially. Second, all combinations of comparison views and target views in an aggregate view are not comparable semantically. We discuss the problem, challenges, and baseline solution of reference dataset refinement in detail in Section 5.2.

Moreover, we design and implement a holistic prototype system View-360 in Python, which included all aspects of aggregate view recommendation i.e, recommendation based on categorical and numerical attributes moreover recommendation based on refinement on target and comparison queries. We also introduce some additional data exploration features in View-360 that help discovery of insights. The effectiveness of our schemes and the proposed prototype system is demonstrated through analysis on multiple datasets. Specifically, user specifies the dataset and set the required parameters. View-360 automatically performs all the steps of view generation and refinements and recommends the top-k ranked views. Figure 5.1 shows the output generated by View-360.

5.2 Reference Dataset Refinement

A common operation during data exploration is exploring subsets of data by progressively changing filters in the WHERE clause of the input query. However, manually changing the input query is a major bottle neck in the exploration process. Consequently, in Chapter 4, we presented schemes for automatic query refinement for the purpose of exploring all subsets of data for aggregate view recommendation. Particularly, the main idea of these schemes is to generate refined queries in order of their similarity with the input query and then generate aggregate views.

As mentioned earlier the recommended aggregate views are based on the target views defined on subsets of data selected by the refined queries and comparison views defined on the complete dataset. The interesting views revealed in such a manner can be considered as *global interesting views*, as the comparison view comes from the complete data (global). However, contrary to global interesting views there can exist *local interesting view*, in which the comparison views come from another subset of data instead of the complete dataset. For example, a sales manager might want to analyze interesting views about sales in each month compared to sales of the whole year, this would be a global interesting view search. However, she might need to analyze sales of months compared to each other, in that case it will be a local interesting view search and the comparison view should also come from subset of data corresponding to a month.

To the best of our knowledge, none of the existing view recommendation systems can recommend the local interesting views. However, some recent efforts have initiated research in this direction. Particularly, the recent work on data slice search [52, 53], has adopted the local outlier detection strategies from the context of knowledge discovery [99]. Specifically, [99] proposes the idea of local outlier factor (LOF), which is based on local density in multi-dimensional space. For each data point, they compute LOF value that indicates the outlierness among its nearest neighbors. The LOF value of data point is high if its local density is low and those of its nearest neighbors are high. In the context of local interesting aggregate views, the idea is to find aggregate views that show interestingness in terms of deviation between a target view which can be analogous to a point in LOF and a comparison view which comes from the neighborhood of the target view. However, the definition of neighborhood of a target view is a non-trivial task, as it involves semantic of the view.

Recently, VisPilot [51], another visual data exploration tool has been developed, which explores the subsets of data and recommends a small set of interesting visualizations to convey key distributions. Similar to OLAP cube they build a lattice of data subsets, where the first node represents the complete dataset. However, in our problem settings data subsets are considered in order of similarity with the input query. Nevertheless, for the purpose of recommending local interesting views a lattice approach can be adopted such that the first node is the input query and then the nodes down the lattice represent refined queries.

5.2.1 View Recommendation with Reference Dataset Refinement

We have been assuming uptill now that for comparison views the reference dataset is predefined by the user by using one of the following three choice: 1) the user provides a second input query to select the referenece dataset, 2) the complete database is used as reference dataset by default, or 3) the complemet of the data selected by input query Q is selected as reference dataset (i.e., reference dataset = $D_B - D_Q$). The target and comparison view should have the same A, M and F. As the deviation is calculated between the probability distributions of the target and comparison views, and the visualization is plotted using those distributions. Having different dimension (A), measure (M) or function (F) will loose the semantics of the visualization and deviation value.

However, the aggregate views recommended by using the specified reference dataset may not be surprising or insightful due to the similarity between the probability distributions of the target and comparison views. Therefore, we propose to refine the reference dataset for generating comparison views instead of using the specified reference dataset. We propose to use the same subsets for comparison as well. However, generating comparison views from refinement of reference dataset, target views from refinement of input query and then combining them in aggregate views form a combinatorial problem. Where more importantly, we need to identify the meaningful combinations of target and comparison views as there will be many combinations in which comparison and target views will lack *contextual reference* to each other. In other words, in such scenario, there exist many aggregate view which have no semantic value for the user.

For instance, consider CENSUS INCOME dataset [20] again and assume for a particular target and comparison view the predicates are:

Target: `WHERE gender == Female and edu` ≥ 12,

Comparison: `WHERE gender == Male and edu` ≥ 7.

Here the female having education higher than high school are compared with male having education higher than primary school. The comparison has no contextual reference as the subsets under comparison are not related in any way. This is because the predicates for target and comparison have completely different values. However, if the predicates have similarity, that will set the context of comparison and consequently, the comparison would make sense. For instance, for the above example, it makes more sense if the comparison is between male and female having education higher than high school or female having education higher than high school compared with females having education higher than primary school as specified by following predicates:

Comparison: `WHERE gender == Male and edu` ≥ 12.

OR

Comparison: `WHERE gender == Female and edu` ≥ 7.

Problem Definition

In a nutshell, Our goal is to help analysts discover the key insights in a dataset by refining the reference dataset and input query. Specifically, this task boils down to generating the set of all refined queries

$\mathbb{Q}$ and then generating the aggregate views. Particularly, for aggregate views, the target views are generate on a subset of data defined by a refined query $Q_j \in \mathbb{Q}$ and the comparison views are generated on another subset of data defined by another refined query $Q_k \in \mathbb{Q}$ such that Q_j and Q_k have contextual reference to each other.

Recall in Section 4.3.2, a query is defined by its predicates ($Q_j = P_{j_0}, P_{j_1}, ...P_{j_n}$). We define a metric to determine the contextual reference between two queries Q_j and Q_k in terms of difference of predicates between the two queries. Particularly, a context is set if the two queries are not different from each other by more than one predicate. Note that, contrary to the case of input query refinement here we are not interested in exact amount of difference of value in predicates. For the sake of knowing that the queries are contextually related, it is enough to know the two queries differ from each other by only one predicate.

$C(Q_j, Q_k) = |Q_j \cap Q_k|$

We define an aggregate view $V_i(Q_j, Q_k)$ as the i^{th} aggregate view from two subsets defined by queries Q_j and Q_k. Specifically, the target view of $V_i(Q_j, Q_k)$ is defined on the subset of data selected by Q_j, while the comparison view is defined on subset selected by Q_k. We define C as a constraint in our problem setting. The utility of a view U is still the deviation based utility define in Chapter 4. Formally, the problem of reference view refinement is stated as:

Definition: Reference View Refinement for View Recommendation: *G*iven an user-specified query Q on a database D_B, a set of refined queries $\mathbb{Q}$, and a multi-objective utility function U, Find k locally interesting aggregate views $V_i(Q_j, Q_k)$ that have the highest utility values, from all of the refined queries $Q_j \in \mathbb{Q}$, such that $C(Q_j, Q_k) = 1$.

All combinations of Q_j and Q_k forms a combinotorial problem. Particularly, let the number of refined target views be N then considering the same views for comparison the possible aggregate views will be $N \times (N-1)$. However, it is not a performance bottleneck for us, as the subsets of data defined by refined queries and the aggregate queries for target views are already being retrieved from the database, therefore, for reference dataset refinement and comparison views the already retried data can be used. However, calculating the constraint C for huge combination of views is computationally expensive. A simple baseline solution is to consider all combinations of Q_j and Q_k and the ones that pass the constraint are used to generate and recommend the top-k views.

5.3 View-360

We present prototype of a visual data exploration tool, titled View-360, that recommends interesting visualizations by espousing the following four aspects: i) automatic binning of numerical dimensions, ii) automatic refinement of input query and reference dataset, iii) ensuring statistical significance of recommended views by hypothesis testing and iv) making sure the recommended aggregate views have semantic value by carefully selecting the target and comparison view for each visualizations.

Moreover, it is unclear how to decide which attributes to use for predicates, dimensions and measures. Therefore, to keep the system flexible and have maximize discovery of insights, we allow to have overlap in set $\mathbb{P}$, $\mathbb{A}$ and $\mathbb{M}$. However, for a view an attribute can be used either as a measure or dimension or in a predicate at one time to keep the semantics of visualizations in tact. For instance, assume attribute `edu` is marked in for all of the sets $\mathbb{P}$, $\mathbb{A}$ and $\mathbb{M}$ by the user. Now when it is used in refinement then it does not make sense to also use it in dimension and perform binning on it or to use it as a measure and perform aggregate function on it, therefore for views that have `edu` as predicate it is removed from $\mathbb{A}$ and $\mathbb{M}$ sets. Later, it is included in set $\mathbb{A}$ and consequently it is removed from the other two sets and so on. In this way it gets to be treated as a dimension, measure and predicate for discovery of insights.

We have also explored the possibilities of diversifying or unifying results and it was discovered that unifying results i.e. putting the related views together is helpful for the user for drawing insights into the data. Therefore, after generating top-k views, View-30 provides three option of viewing the views: 1) in order of their ranking, 2) display the diversified top-k views, and 3) the results from the same target query are put together or the results having the same dimension attribute are put together. The analyst can perform analysis using one or all of the three options mentioned above. This helps to synthesizing insights into data. We discuss this in detail in our analysis of various datasets.

Moreover, View-360 also provides a feature to explore a view further by plotting the scatter plot or the frequency distribution of the attributes involved in that view. This feature is extremely useful in many cases as it is shown in the detail discussion of datasets in next sections. Particularly, when a recommended view has `SUM` as aggregate function, the explanation of such a view is not as straightforward as other aggregate functions. For instance, the high value of `SUM` of a measure attribute for a particular category in dimension attribute can be due to two reasons: 1) the value of the measure itself is high, or 2) the value of the measure attribute is small, but the number of instances in that category are large and when they add up for the SUM function it becomes a large value. Moreover, if the high value is due to (1), then the same view with `COUNT` as aggregate function should also has the ranking close to the one with the `SUM` function. Therefore, to draw an insight from a view that has the `SUM` aggregate function, the same view with `COUNT` aggregate function should also be looked at. View-360 further exploration feature, provides this facility to quickly look at the required related view or other visualizations that helps in explanation of insight.

5.3.1 General Settings

In default settings, we assume there is no input query, therefore, all possible input queries (subsets of data) are considered for recommendation.The user specifies attributes from the dataset for the following parameters:

$\mathbb{A}_c$ - Categorical dimension attributes

$\mathbb{A}_n$ - Numerical dimension attributes

$\mathbb{M}$ - Measure attributes

$\mathbb{P}_c$ -Categorical attributes for predicates

$\mathbb{P}_n$ - Numerical attributes for predicates

$\mathbb{F}$ - Aggregate functions

k - k for top-k

For attributes in $\mathbb{P}_c$, refinements are generated by all unique categories of each attribute, for attributes in $\mathbb{P}_n$, user specified discritization factor (γ) is used to discretize the continuous values and then all possible refined queries are generated. Each refined query can be used in the target view and comparison view. For each refined query views are generated with all combinations of $\mathbb{A}$, $\mathbb{M}$ and $\mathbb{F}$. The refinement is applied to categorical attributes as well, however these are not used in conjunction with each other. Specifically, only one categorical attribute is used at a time in refinement.

5.4 Business Domain

5.4.1 Flight Delays and Cancellations 2015

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights is published in DOT's monthly Air Travel Consumer Report. In this analysis we use the dataset of 2015 flight delays and cancellations [94]. The attributes of the dataset are described in the Table 5.1.

5.4.2 Data Pre-processing

The flights dataset [94] consist of three tables i.e., `Flights`, `Airports` and `Airlines`. The flights table has 5,819,079 flight records, which are described by 32 attributes. `Airports` and `Airlines` tables have details about the airports and airlines, respectively.

Table 5.1 shows that there are missing values in attributes related to flight-delay-reasoning such as `AIRLINE DELAY, WEATHER DELAY`. Therefore those attributes are removed from analysis. Table 5.1 also shows that the `CANCELLATION REASON` attribute has 98.4% missing values, this is because this attribute is only filled for the flights that are canceled (which are very low in number). Therefore, we keep this attribute and in the next section we explain how we use this attribute in analysis. Note that all other attributes are more than 98% complete.

Moreover, the attributes `DAY OF WEEK` and `MONTH` are converted to equivalent text for readability of results.

In Table 5.2, the number of categories for each nominal attribute are shown. Table 5.2 shows the `YEAR` attribute has only one value, because the dataset is only for the year 2015, therefore we remove the attribute `YEAR` from further processing.

In the dataset, the unique values of airport codes in `ORIGIN AIRPORT` and `DESTINATION AIRPORT` attributes were more than the lookup values in the `Airports` table. Numeric values existed which did not match with the `Airports` table. Consequently, the numeric airport codes are removed by joining `Flights` with `Airports` table. Eventually, a single table is created which contains valid airport codes

Attribute Name	Type	Description & Values	% missing
YEAR	Numeric	Year of the Flight Trip	
MONTH	Numeric	Month of the Flight Trip	0%
DAY	Numeric	Day of the Flight Trip	0%
DAY OF WEEK	Numeric	Day of week of the Flight Trip	0%
AIRLINE	Nominal	Airline Identifier	0%
FLIGHT NUMBER	Numeric	Flight Identifier	0%
TAIL NUMBER	Nominal	Aircraft Identifier	0.2%
ORIGIN AIRPORT	Nominal	Starting Airport	0%
DESTINATION AIRPORT	Nominal	Destination Airport	0%
SCHEDULED DEPARTURE	Numeric	Planned Departure Time	0%
DEPARTURE TIME	Numeric	WHEEL OFF - TAXI OUT	1.5%
DEPARTURE DELAY	Numeric	Total Delay on Departure	1.5%
TAXI OUT	Numeric	The time duration elapsed between departure from the origin airport gate and wheels off	1.5%
WHEELS OFF	Numeric	The time point that the aircraft's wheels leave the ground	1.5%
SCHEDULED TIME	Numeric	Planned time amount needed for the flight trip	0%
ELAPSED TIME	Numeric	AIR TIME+TAXI IN+TAXI OUT	1.8%
AIR TIME	Numeric	The time duration between wheels off and wheels on time	1.8%
DISTANCE	Numeric	Distance between two airports	0%
WHEELS ON	Numeric	The time point that the aircraft's wheels touch on the ground	1.6%
TAXI IN	Numeric	The time duration elapsed between wheels-on and gate arrival at the destination airport	1.6%
SCHEDULED ARRIVAL	Numeric	Planned arrival time	0%
ARRIVAL TIME	Numeric	WHEELS ON+TAXI IN	1.5%
ARRIVAL DELAY	Numeric	ARRIVAL TIME-SCHEDULED ARRIVAL	1.8%
DIVERTED	Numeric	Aircraft landed on airport that out of schedule	0%
CANCELLED	Numeric	Flight Cancelled (1 = cancelled)	0%
CANCELLATION REASON	Nominal	Reason for Cancellation of flight: A - Airline/Carrier; B - Weather; C - National Air System; D - Security	98.4%
AIR SYSTEM DELAY	Numeric	Delay caused by air system	81.7%
SECURITY DELAY	Numeric	Delay caused by security	81.7%
AIRLINE DELAY	Numeric	Delay caused by the airline	81.7%
LATE AIRCRAFT DELAY	Numeric	Delay caused by aircraft	81.7%
WEATHER DELAY	Numeric	Delay caused by weather	81.7%

Table 5.1: Flights Delay Dataset: Attributes Description

and airport names. Additionally, `Flights` table was joined with the `Airline` table as well to get the names and location of airports instead of just codes.

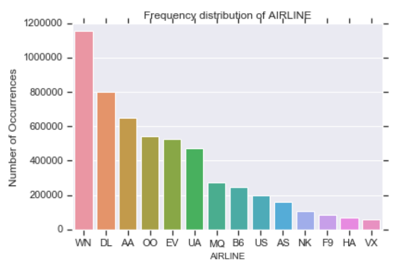
The `ARRIVAL DELAY` and `DEPARTURE DELAY` columns have negative values which show early departure and arrival time. However, for analysis any values that are less than zero are considered as no delay. Secondly, `DEPARTURE DELAY` is a very important column of the flight delay dataset, View-360 uses it as a dimension attribute. However, the automatic binning of View-360 consider equal width bins only. For adding more depth to the analysis, a new categorical calculated attribute is added for the departure delay values with unequal size fixed bins. Particularly, if the departure delay is less than 5 minutes then the flight is considered on time. If the departure delay is between 5 to 45 minutes, it is consider as a small delay and all delays more than 45 minutes are considered as large delays. This attribute is named as `DEPARTURE DELAY C`.

Attributes Distribution

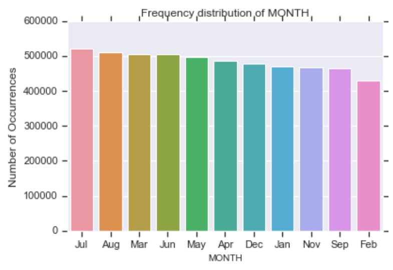
As a pre-processing step, the frequency distributions of various attributes are plotted, as shown in Figure 5.2. Firstly, these frequency distributions show that the categories in attribute `AIRLINE, MONTH, DAY OF WEEK, DEPARTURE DELAY` are reasonably represented. Secondly, as the aggregate views recommended by View-360 are based on distance between probability distributions of target view and comparison view, sometimes for explanation of the insight in recommended visualization, it

Attribute Name	Number of Unique Values
YEAR	1
MONTH	12
DAY	31
DAY OF WEEK	7
AIRLINE	14
ORIGIN AIRPORT	628
DESTINATION AIRPORT	629
DIVERTED	2
CANCELLED	2
CANCELLATION REASON	5

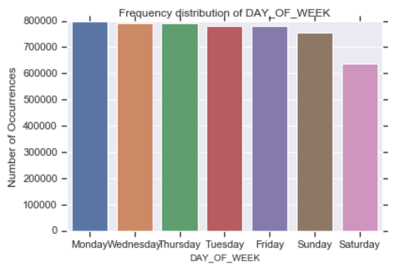
Table 5.2: Flights Delay Dataset: Nominal attributes



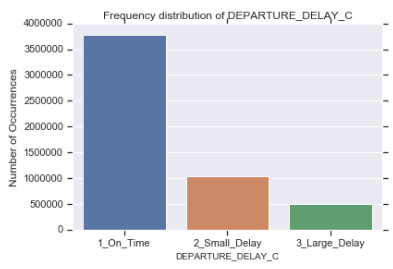
(a) Airline



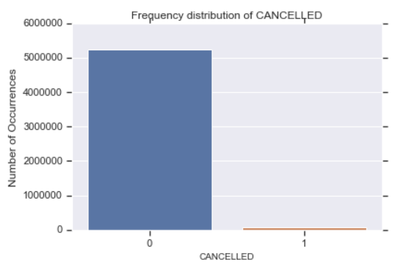
(b) Month



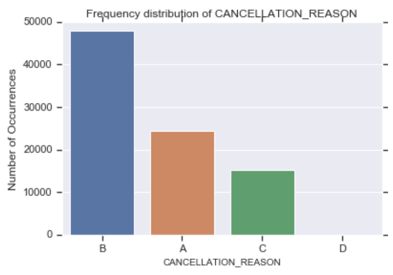
(c) Day of Week



(d) Departure Delay



(e) Cancelled



(f) Cancellation Reason

Figure 5.2: Flight Delays Dataset: Attributes Distribution

is helpful to look at the frequency distribution of the attributes involved in the visualization.

Moreover, Figure 5.2e and 5.2f show that very few flights are canceled and out of those most cancellations are due to reason 'B' which corresponds to weather related issues. Therefore, canceled flights are not analyzed in the View-360 and the focus of analysis is on the delayed flights.

5.4.3 Experiments Settings

As mentioned, in Section 5.3, in View-360, overlapping sets of attributes can be specified as input parameters. Accordingly, the dimensions, measures and aggregate functions in View-360 are set as follows:

- $\mathbb{A}_c$: `MONTH, DAY OF WEEK, AIRLINE, DESTINATION AIRPORT, ORIGIN AIRPORT, DEPARTURE DELAY C`
- $\mathbb{A}_c$: `AIR TIME, DISTANCE, ARRIVAL DELAY, DEPARTURE DELAY`
- $\mathbb{F}$: Count, Mean, Sum

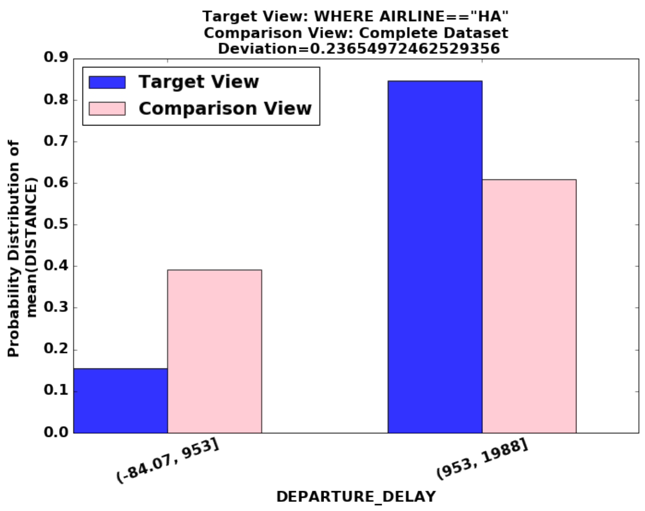


Figure 5.3: Flight Delays Dataset: Hawaiian Airline (V_1)

- $\mathbb{M}$: AIR TIME, DISTANCE, ARRIVAL DELAY, DEPARTURE DELAY
- $\mathbb{P}_n$: AIR TIME, DISTANCE, ARRIVAL DELAY, DEPARTURE DELAY, SCHEDULED DEPARTURE
- $\mathbb{P}_c$: MONTH, DAY OF WEEK, AIRLINE, DESTINATION AIRPORT,

  ORIGIN AIRPORT, DEPARTURE DELAY C
- k: 50

The input query Q is set to null, which means consider all possible refinements on attributes of set P_n and P_c. This scenario is particularly useful when the analyst has no preferred data to start the exploration process. Moreover, in the absence of input query, the similarity objective of the multi-objective function in QuRVe becomes irrelevant. Therefore, $\alpha_S = 0$ and $\alpha_D = 1$.

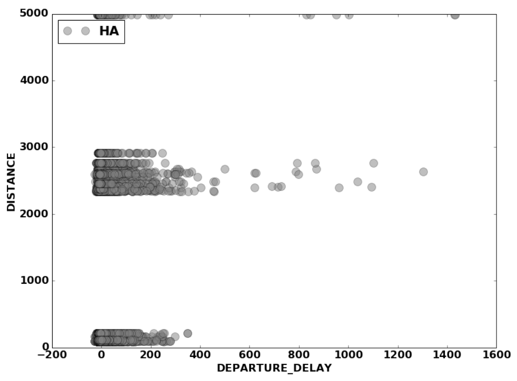
5.4.4 Summary of Results

View-360 employs QuRVe scheme to generate the set $\mathbb{Q}$, of all of the refined queries using the attributes specified by P_n and P_c. Then all aggregate views for each $Q_j \in \mathbb{Q}$ are generated and their deviation-based utility is computed. Furthermore, for the numerical attributes of set A, MuVE scheme is employed by View-360 that perform automatic binning and generate all binned views. As mentioned in Section 5.3, View-360 displays the top-k views and then provides the option of diversifying or unifying the views. Particularly, for unifying the results it combines the views with respect to refined queries and/or dimensions of the view and then display the combinations. For this analysis the top-k views are combined by their underlying refined queries.

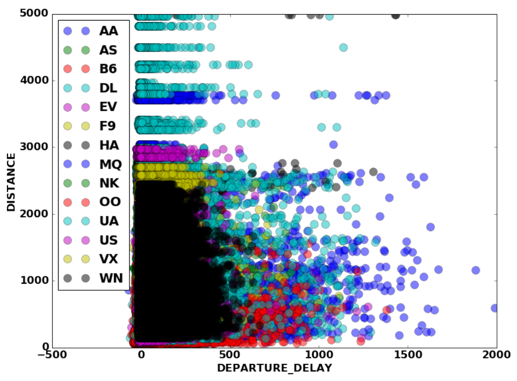
Analysis 1: Hawaiian Airline

This analysis is based on a refined query Q_j which appeared in 7 of the top-10 views recommended by View-360. The refined query selects the subset of data for *HA=Hawaiian* Airline i.e.,

Q_j: SELECT * FROM Flight-delays WHERE Airline=='HA'



(a) Hawaiian Airline



(b) All Airlines

Figure 5.4: Departure delay vs. Distance

Figures 5.3, 5.5, and 5.7 show three interesting views out of those seven views. As mentioned earlier that View-360 employ QuRVe scheme to automatically refine the queries and MuVE scheme for automatically binning the numerical dimensions. Hence, through QuRVe the above mentioned refine query is discovered and through binning the best binning options that expose insights are discovered. For all views shown in Figures 5.3, 5.5, and- 5.7, the target view is on the subset of data that represents flights from Hawaiian airline only, while the comparison view is on the data of all airlines.

In Figure 5.3 on the x-axis the numerical dimension `DEPARTURE DELAY` is shown. As it is a numerical dimesion attribute, therefore, it is automatically binned by MuVE. The view shown in Figure 5.3 is binned into two equal width bins. In particular, the first bin is between -84.07-953 minutes and second bin is between 954-1988 minutes. On the y-axis Figure 5.3 shows the probability distribution of `MEAN` of the `DISTANCE` covered by flights for comparison and target views. The comparison view in Figure 5.3 shows that the flights having departure delay more than 953 minutes have 60% mean distance as compared to their counter part which have 40% mean distance. However, the target view in Figure 5.3 shows that the flights that have a departure delay less than 953 minutes have less than 20% of mean distance covered,while the flights that have departure delay greater than 953 minutes have more than 80% of mean distance. This implies that generally heavy departure delay can be experienced for log distance flights in case of Hawaiian Airline. To investigate this further, View-360's further exploration feature is used, and scatter plot of `DEPARTURE DELAY` vs. `DISTANCE` for Hawaiian Airline and for all airlines is generated as shown in Figure 5.4a and Figure 5.4b respectively. Figure 5.4 shows that Hawaiian airline has number of flights that have maximum distance and departure delay more than 953 minutes while for all other airlines there are not many flight in that area of scatter plot. This confirms our insight that the long distance flights from Hawaiian airline have high chances of experiencing long departure delay. Furthermore, it is expected that when the flights are delayed at departures, the same amount of delay will be recorded at the arrival. Figure 5.5 shows another recommended view by View-360, which has the same underlying refined query as the last one. Specifically, the view shown in Figure 5.5 has the categorical departure delay attribute i.e., `DEPARTURE DELAY C` as dimension attribute on the x-axis. Recall for this attribute if the departure delay is less than 5 minutes then the flight is considered on time. If the departure delay is between 5 to 45 minutes,

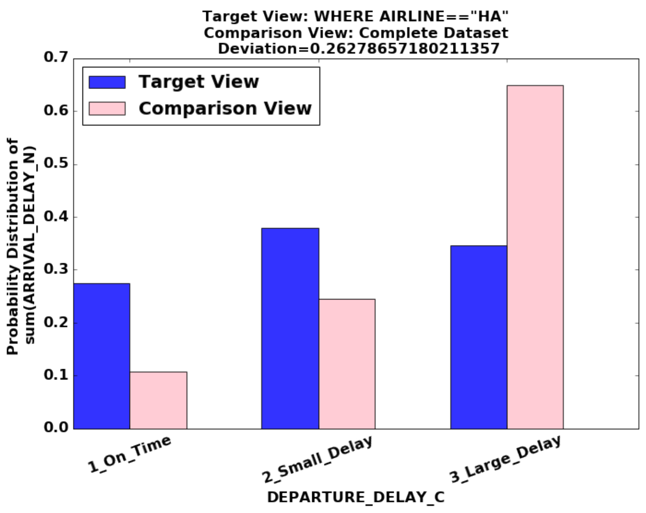


Figure 5.5: Flight Delays Dataset: Hawaiian Airline (V_2)

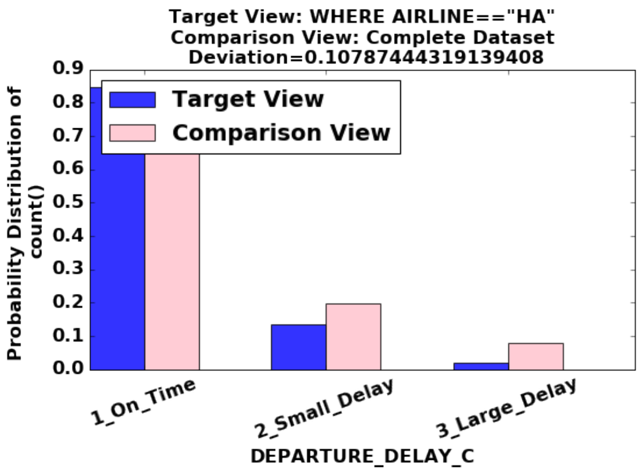


Figure 5.6: Flights Distribution

we consider it small delay and all delays more than 45 minutes are considered as large delays, these categories are mentioned in the previous section. On the y-axis the Figure 5.5 shows probability distribution of `SUM` of `ARRIVAL DELAY`. The comparison view shows the probability distribution of the sum of arrival delay for all airlines corresponding to each departure delay category. The probability distribution of comparison view shows that as the departure delay increases, the sum of arrival delay also increases. However, in Figure 5.5, the target view for flights corresponding to *HA=Hawaiian* Airlines shows a different trend. For Hawaiian Airline the probability distribution of the sum of the arrival delay is almost the same for all categories of the departure delay.

For instance, in the aggregate view of Figure 5.5, the probability distribution in the target view, either imply that Hawaiian airline has a different number of flights for departure delay categories compared to the comparison view, or Hawaiian Airline has the shown trend due to the sum of the arrival delay. In other words, Hawaiian airline incur arrival delays irrespective of how much departure delay was experience by the flight. If the former was true then a view with `COUNT` aggregate function should also have come up in the top-k, but that is not the case. However, using View-360 further exploration is performed and the view with `COUNT` as the aggregate function in Figure 5.6 is examined. The views confirms our analysis that the Hawaiian airline has similar pattern as for all other airlines

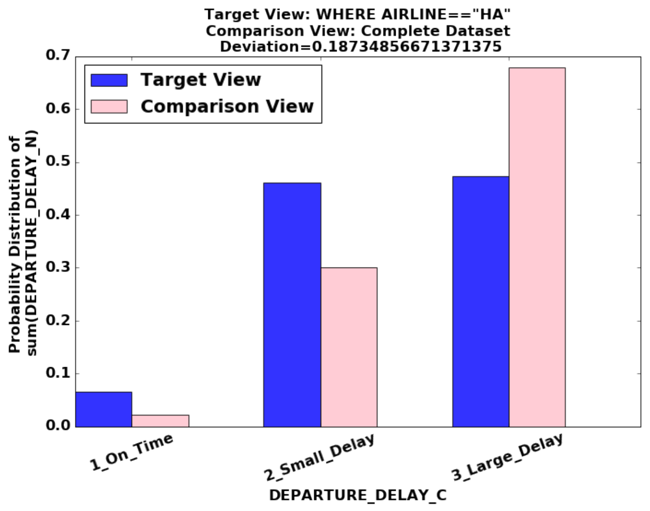


Figure 5.7: Flight Delays Dataset: Hawaiian Airline (V_3)

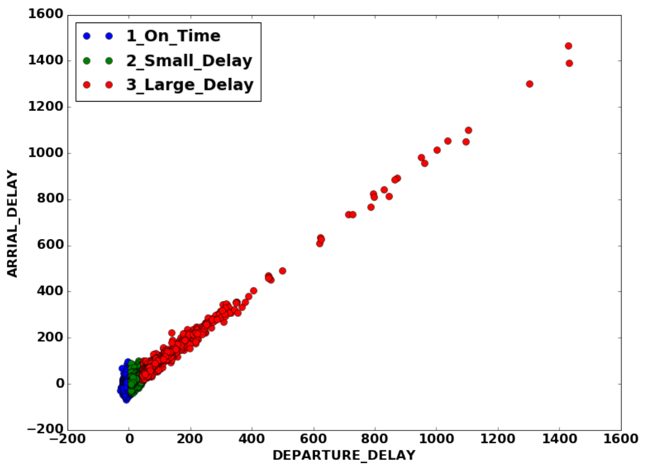
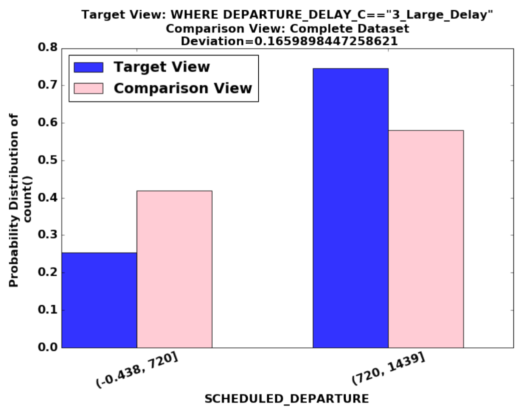


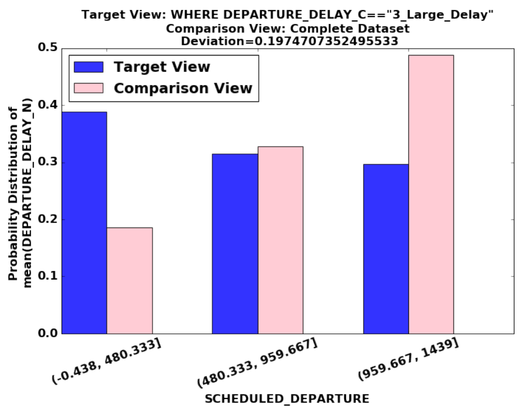
Figure 5.8: Departure Delay vs. Arrival Delay for Hawaiian Airline

for the number of flights. Therefore, we believe that Hawaiian airline has a pattern of unusual arrival delay.

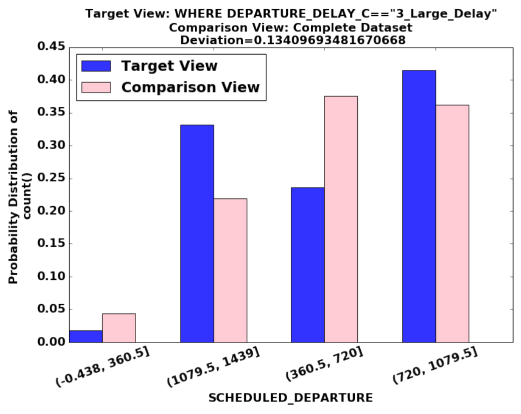
Figure 5.7 shows the probability distribution of SUM of DEPARTURE DELAY attribute on the y-axis. Figure 5.7 also reassures the insight of the Figure 5.5 for Hawaiian airline. Specifically, it can be clearly seen that in the target view the percentage of sum departure delay of flights that were on time is less than 10%, while in target view of Figure 5.5 the sum of arrival delay percentage for on time category it is almost 30%. This can be further explained from Figure 5.8, where the departure delay and arrival delay for Hawaiian airline is plotted by View-360 as a scatter plot,which shows Hawaiian airline incur arrival delay even for on time and small departure delay flights. Moreover, it can also be seen that the sum of arrival delay for comparison view in Figure 5.5 is comparatively less than the sum of departure delay in Figure 5.7 for flights that experienced high departure delays. This indicates that sometimes airlines adjusted their flight in order to reduce the delays at arrival.



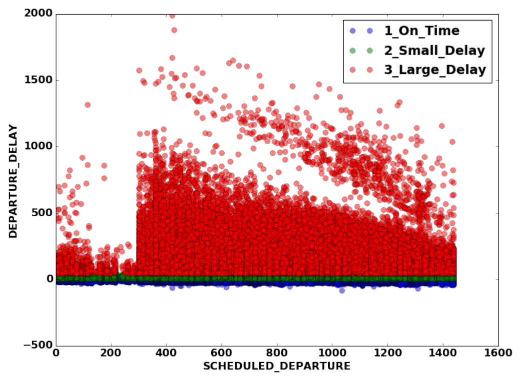
(a)



(b)



(c)



(d)

Figure 5.9: Scheduled Departure Time vs. Departure Delay

Analysis 2: Scheduled Departure Time vs. Departure Delay

This particular analysis is related to the refined query generated by QuRVe, which selects the flights that have departure delay more than 45 minutes i.e.,

`SELECT * FROM Flight-delays WHERE Departure_Delay_C=='3_Large_Delay'`

The dimension that stands out in the recommended of View-360, particular to this refined query is the SCHEDULED DEPARTURE, which is the time for which the flight was scheduled to depart. The views in Figure 5.9 are 3 of the top-k views corresponding to above mentioned refined query and has scheduled time of flights on the x-axis. All of these views are binned views and are discovered by View-360 by employing MuVE. Particularly, Figure 5.9a shows a view in which the scheduled departure time is binned into 2 equi-width bins, i.e., 12 hours in each bin, while the y-axis has the probability distribution of COUNT. The comparison view in the Figure 5.9a shows around 40% of the flights are scheduled for first half of the day while around 60% are scheduled for the second half of the day. However, in target view which corresponds to flights with departure delay greater than 45 minutes, it clearly shows that almost $\frac{3}{4}th$ of the flights are delayed in the second half of the day.

Figure 5.9c shows a deeper insight into the same situation. Here the number of bins are 4, which means the day is divided into intervals of 6 hours i.e., 360 minutes. The comparison view in Figure 5.9c shows that for the first bin i.e., 12:00 am-6:00 am onlt 5% of the flights are scheduled. Then for the middle two intervals i.e., 6:00 am-6:00 pm almost equal number of flights are scheduled and this is the loaded time of the day having 70% of all flights. For the last interval i.e., 6:00 pm-12:00am the number of flights decreases and comes down to 25%. The target view in the Figure 5.9c shows that the highest number of flights that experienced large delayed is between 12:00 pm- 6:00 pm.

Figure 5.9b reveals another aspect of the scenario under discussion. On the x-axis the SCHEDULED DEPARTURE time is binned into 3 equi-width bins and on the y-axis the probability distribution of the MEAN of DEPARTURE DELAY is shown. The comparison view in the figure shows that mean departure delay of all the flights is highest for the later part of the day. However, the target view shows that for the flights experienced large delays, almost the 40% of the mean departure delay is between 12:00 am 8:am. We already know during this time slot very few flights are scheduled, however, the view suggests that the flights that get delayed in this slot experience big departure delays. To confirm this insight, View-360's further exploration feature is used and the scatter plot between SCHEDULED DEPARTURE time and DEPARTURE DELAY is generated, as shown in Figure 5.9d. It can be clearly seen in the plot that between 0-4800 i.e., 12:00 am - 8:00 am the departure delay is highest as compared to any other time of the day.

Analysis3: Busy Airports

There are 322 unique airport codes in the dataset, it is not feasible to plot all the unique values on the x-axis for views. Secondly, as it is a categorical attribute therefore automatic binning can not be performed. The airports can be grouped together in a hierarchy, for instance by city or state. However, this separate set of analysis is performed by selecting a subset of data of highly loaded airports (i.e. the top-15 airports only).

Figure 5.10a shows one of the top-k view recommended by View-360 by employing QuRVe which automatically refined the input query. The Figure 5.10a has codes of the 15 selected ORIGIN AIRPORTS on the x-axis and the probability distribution of COUNT on the y-axis. The comparison view shows probability distribution of flights from all airlines corresponding to each of the busy airports. It can be clearly seen that ATL=Atlantic City International Airport and ORD=Chicago O'Hare International Airport are the busiest airports, together they have 30% of the flights of 15 airports. The Target view shows that the Airline DL=Delta Airlines have almost 50% of their flights from ATL airport. Additionally, from our pre-processing and attribute analysis we already know that DL is actually the second largest airline as shown in Figure 5.2a. Hence, it is learnt that majority of DL flights are from ATL, this is evidence to the fact the DL has it's headquarters in ATL.

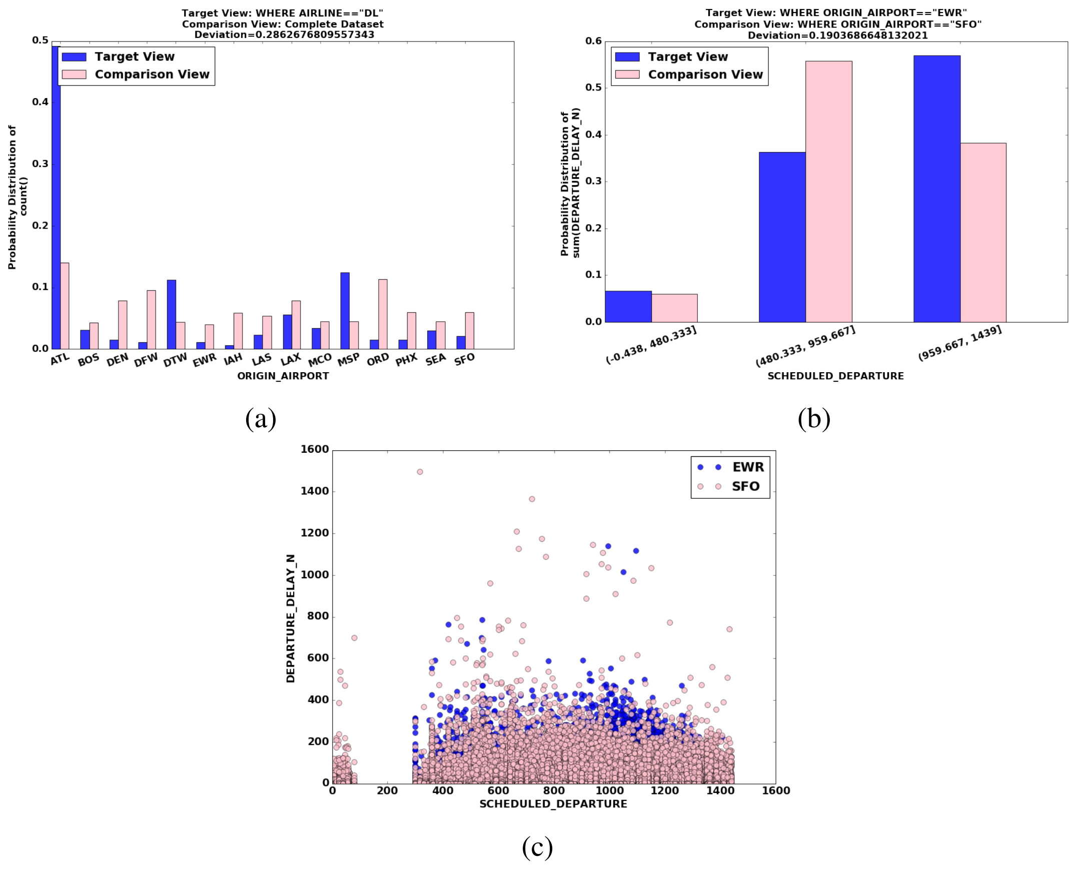


Figure 5.10: Flight Delays Dataset: Busy Airports

Next, View-30 is configured to include the refinement of the reference dataset as well to recommend top-k views. Figure 5.10b shows probability distribution of `SUM` of `DEPARTURE DELAY` for `EWR= Newark Liberty International Airport, NJ` and `SFO=San Francisco International Airport` vs. the `SCHEDULED DEPARTURE` time. This particular view is generated by MuVE through which the scheduled departure time is binned into 3 bins i.e., each bin consists of around 8 hours slot of the day for instance first bin corresponds to 12:00 am-8:00 am or in terms of minutes it is 0-480 minutes. The comparison view is for `SFO` airport, which shows the probability of the sum of departure delay for this airport is minimum for the first slot of the day and it is highest for second bin i.e., 8:00 am - 4:00 pm. However, the target view for the Airport `EWR` shows that the probability of the sum of departure delay for this airport is maximum for the last slot of the day. To investigate further, Figure 5.10c shows scatter plot of scheduled departure time and departure delay for these two airports generated by View-360. It can be clearly seen that for `SFO` flights scheduled between 480-960 minutes, there are many flights with high delays, which explains the comparison view of the Figure 5.10b.

5.5 Health Domain

Healthcare, like many service industries across the globe, is now grappling with how to use the enormous amount of digital data being generated to inform clinical decision in an efficient way [100]. The use of data visualizations is deemed essential to generate useful insights that clinicians and healthcare sector can trust and action in a timely manner . A recent review by Galetsi and Katsaliaki around state of data analytic in healthcare emphasizes the role improvements in real-time data science including visualization techniques have to play in the research of the most severe diseases that humanity faces nowadays such as, cancer, Alzheimer, diabetes, etc., [101].

For diseases like diabetes with high global occurrence, it has been highlighted that there will not be enough family care doctors to meet future needs [102]. Any allied healthcare professionals or carers as physician assistants or family members, will likely take that role, but they will need easy-to-understand visual information tools that give them quick yet accurate insights [24]. Therefore taking a case of using a diabetes dataset, the proposed techniques in this research demonstrate how user-centred visualizations simplify the interpretation process by presenting insights.

Diabetes Mellitus is a disease marked by high levels of sugar in the blood. WHO foresees that diabetes will be the 7th leading cause of death in 2030. In the recent years, the interest in reducing hospital re-admissions has increased due to its potential to reduce healthcare costs and improve care. Particularly, the interest in reducing diabetes hospital re-admissions has increased because of the growth of the burden of diabetes. In order to reduce diabetes re-admissions with interventions, risk factors of re-admissions should be better understood. Owing to the fact that it would be very costly to apply intervention measures to all diabetic patients, studies usually focus on high-risk patients. There are currently several risk factors which predispose to diabetes such as genetics, race, physiological measures and habits [4]. Hence, for health domain we choose to analyze diabetes patients data using our techniques.

5.5.1 Diabetes Patients Dataset

The dataset [103] represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 features representing patient and hospital outcomes. Information was extracted from the database for encounters that satisfied the following criteria.

1. It is an inpatient encounter (a hospital admission).
2. It is a diabetic encounter, that is, one during which any kind of diabetes was entered to the system as a diagnosis.
3. The length of stay was at least 1 day and at most 14 days.
4. Laboratory tests were performed during the encounter.
5. Medications were administered during the encounter.

The attributes of the dataset are described in Table 5.3.

5.5.2 Data Pre-processing

The original dataset contains 1,01,766 records. However there is incomplete information as expected in any real-world data. There were several attributes that could not be used in analysis directly since they

Feature name	Type	Description and values	% missing
Encounter ID	Numeric	Unique identifier of an encounter	0%
Patient number	Numeric	Unique identifier of a patient	0%
Race	Nominal	Values: Caucasian, Asian, African American, Hispanic, and other	2%
Gender	Nominal	Values: male, female, and unknown/invalid	0%
Age	Nominal	Grouped in 10-year intervals: [0, 10), [10, 20), . . ., [90, 100)	0%
Weight	Numeric	Weight in pounds	97%
Admission type	Nominal	Integer identifier corresponding to 9 distinct values	0%
Discharge disposition	Nominal	Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available	0%
Admission source	Nominal	Integer identifier corresponding to 21 distinct values, for example, physician referral, emergency room, and transfer from a hospital	0%
Time in hospital	Numeric	Integer number of days between admission and discharge	0%
Payer code	Nominal	Integer identifier corresponding to 23 distinct values	52%
Medical specialty	Nominal	Integer identifier of a specialty of the admitting physician, corresponding to 84 distinct values	53%
Number of labnprocedures	Numeric	Number of lab tests performed during the encounter	0%
Number of procedures	Numeric	Number of procedures (other than lab tests) performed during the encounter	0%
Number of medications	Numeric	Number of distinct generic names administered during the encounter	0%
Number of outpatient visits	Numeric	Number of outpatient visits of the patient in the year preceding the encounter	0%
Number of emergency visits	Numeric	Number of emergency visits of the patient in the year preceding the encounter	0%
Number of inpatient visits	Numeric	Number of inpatient visits of the patient in the year preceding the encounter	0%
Diagnosis 1	Nominal	The primary diagnosis (coded as first three digits of ICD9); 848 distinct values	0%
Diagnosis 2	Nominal	Secondary diagnosis (coded as first three digits of ICD9); 923 distinct values	0%
Diagnosis 3	Nominal	Additional secondary diagnosis (coded as first three digits of ICD9); 954 distinct values	1%
Number of diagnoses	Numeric	Number of diagnoses entered to the system	0%
Glucose serum test result	Nominal	Indicates the range of the result or if the test was not taken. Values: $> 200, > 300$, "normal," and "none" if not measured	0%
A1c test result	Nominal	Indicates the range of the result or if the test was not taken. Values: > 8 if the result was greater than 8%, > 7 if the result was greater than 7% but less than 8%, "normal" if the result was less than 7%, and "none" if not measured	0%
Change of medications	Nominal	Indicates if there was a change in diabetic medications (either dosage or generic name). Values: "change" and "no change"	0%
Diabetes medications	Nominal	Indicates if there was any diabetic medication prescribed. Values: "yes" and "no"	0%
24 features for medications	Nominal	such as insulin,glyburide-metformin, etc., the feature indicates whether the drug was prescribed or there was a change in the dosage. Values: "up" if the dosage was increased during the encounter, "down" if the dosage was decreased, "steady" if the dosage did not change, and "no" if the drug was not prescribed	0%
Readmitted	Nominal	Days to inpatient readmission. Values: < 30 if the patient was readmitted in less than 30 days, > 30 if the patient was readmitted in more than 30 days, and "No" for no record of readmission	0%

Table 5.3: Diabetes Dataset: Attributes Description [4]

had a high percentage of missing values. These attributes were `Weight` (97% values missing), `Payer code` (40%), and `Medical specialty` (47%). `Weight` attribute was considered to be too sparse and it was not included in further analysis. `Payer code` was also removed since it had a high percentage of missing values and it was not considered relevant to our study.

The attribute `Diagnosis 1` is the primary diagnosis for each encounter and it is coded as first three digits of ICD9. According to ICD9 codes, we have divided the primary diagnosis into five major categories i.e., Respiratory, Circulatory, Injury, Diabetes, Digestive and others. For our first analysis we used the complete dataset. However, as most of the recorded attributes are relevant to the diabetes disease such as the attributes, `Diabetes medicine`, `HbA1c Test` results, therefore for later analysis, we consider the patients with diabetes as primary diagnosis. Only the code 250.xx corresponds to Diabetes mellitus. Therefore, we consider the subset of the dataset which corresponds to the primary

Description	ICD-9-CM code
Diabetes mellitus without mention of complications	250.0x
Diabetes with ketoacidosis	250.1x
Diabetes with hyperosmolarity	250.2x
Diabetes with other coma	250.3x
Diabetes with renal manifestations	250.4x
Diabetes with ophthalmic manifestations	250.5x
Diabetes with neurological manifestation	250.6x
Diabetes with peripheral circulatory disorders	250.7x
Diabetes with other specified manifestations	250.8x
Diabetes with unspecified complications	250.9x
Diabetes – not stated as uncontrolled	250.x0 or 250.x1
Diabetes – uncontrolled	250.x2 or 250.x3

Table 5.4: Diabetes Dataset: ICD-9-CM Diagnosis Codes [5]

diagnosis as Diabetes mellitus. This results in 8,757 records.

Attributes Distribution

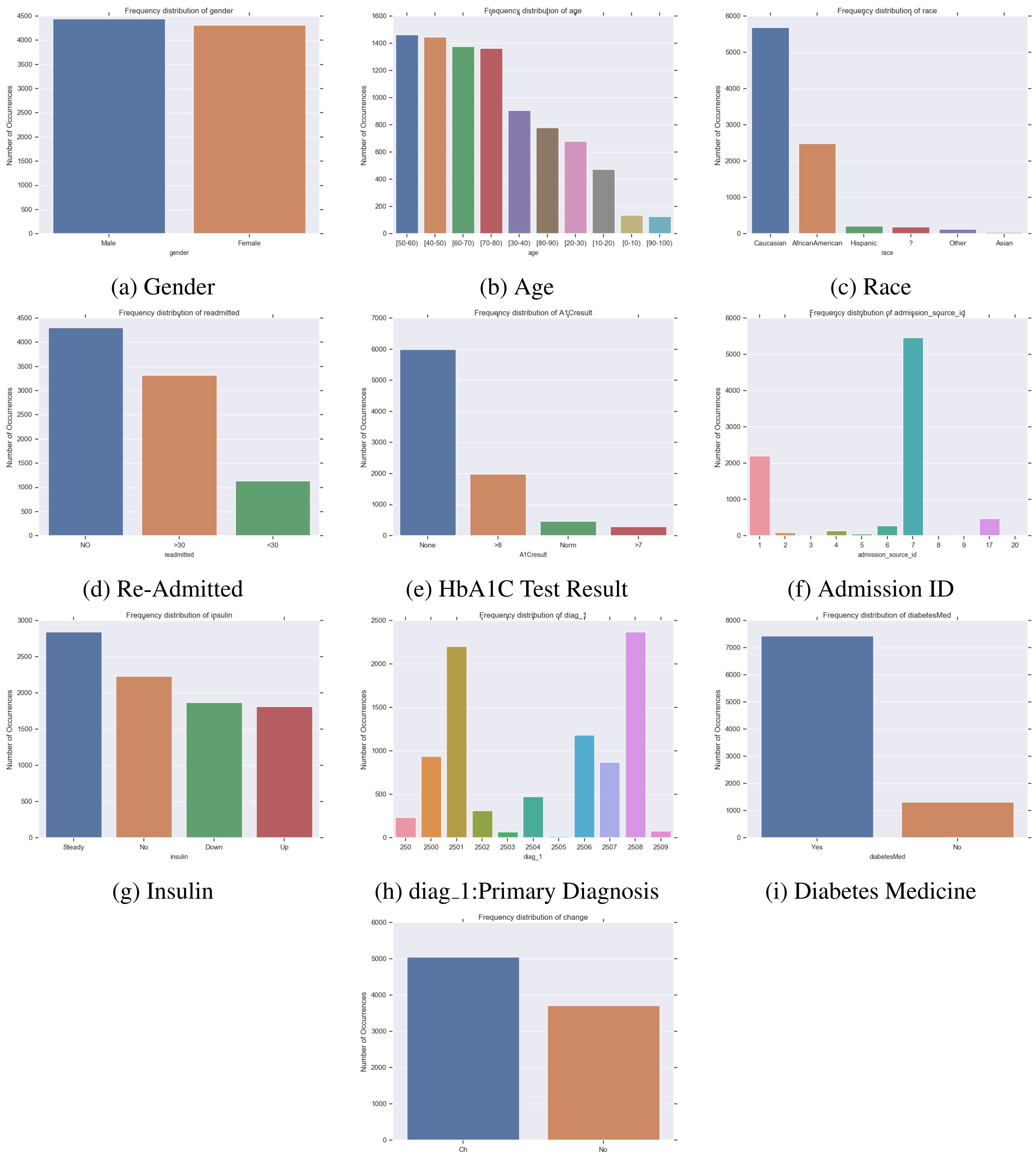
As a pre-processing step, the frequency distribution of various attributes are shown in Figure 5.11 to identify the skewed attributes. These distributions are also readily available in View-360 in further exploration feature of the recommended views. Figure 5.11c shows that the `Race` attribute has extreme representation of two categories while the other categories are in very low representation. However, as it is an important attribute and may show some interesting insights therefore, the `Race` attribute is kept in the analysis and the significance testing takes care of the low representation categories. Figure 5.11f shows that the attribute `Admission ID` has 11 unique categories but only 2 has reasonable representation. This attribute is excluded from analysis due to not enough representation of all categories. The attributes available to control for patient demographic and illness severity were `Gender`, `Age`, `Admission source`, `Discharge disposition`, `Primary diagnosis`, and `Time in hospital`. Additionally, the attributes `Re-admissions`, `HbA1c test` result, `Insuline`, and `Diabetes medicine` are also included in the analysis.

Primary diagnosis used ICD-9 diagnosis codes, Table 5.4 shows the details of the codes.

5.5.3 Experiment Settings

We set our dimensions, measures and aggregate functions as follows:

- $\mathbb{A}_c$: `Age, Race, Gender, Admission Source, Re-admitted, HbA1C test result, Insulin, Diabetes Medicine, Change in Medicine, Discharge Disposition`
- $\mathbb{F}$: Count, Mean, Sum
- $\mathbb{M}$: `Number of lab procedures, Number of procedures, Number of medications, Number of outpatient, Number of emergency, Number of inpatient, Days in Hospital`
- $\mathbb{P}_n$: `Days in Hospital, Number of emergency`



(a) Gender (b) Age (c) Race

(d) Re-Admitted (e) HbA1C Test Result (f) Admission ID

(g) Insulin (h) diag_1:Primary Diagnosis (i) Diabetes Medicine

(j) Change in Medicine

Figure 5.11: Diabetes Dataset: Attributes Distribution

- $\mathbb{P}_c$: `Age, Race, Gender, Admission Source, Discharge Disposition, HbA1C test result, Insulin, Diabetes Medicine, Change in Medicine, Re-admitted`
- k: 100

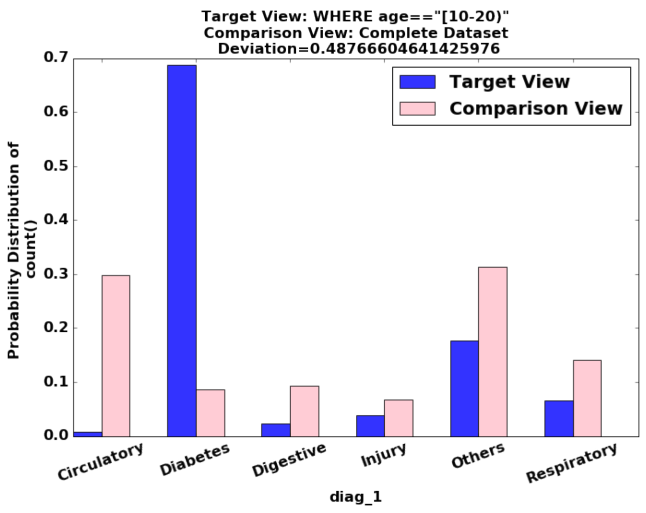


Figure 5.12: Diabetes Dataset: Age Group `[10-20)` V_1

5.5.4 Summary of Results

As mentioned in previous section, View-360 employs QuRVe scheme to generate the set $\mathbb{Q}$, of all of the refined queries using the attributes specified by P_n and P_c. Then all aggregate views for each $Q_j \in \mathbb{Q}$ are generated and their deviation-based utility is computed. Furthermore, for the numerical attributes of set A, MuVE scheme is employed by View-360 that perform automatic binning and generate all binned views. Moreover, for this analysis, View-360 is configured to combine and display the top-k views by their underlying refined queries.

Analysis 1: All Diseases

After unification the top-1 query recommended by View-360 by employing QuRVe scheme to generate the refined queries, selects the patients in age group `'[10-20)'`.

```
SELECT * FROM Diabetes WHERE Age=='[10-20)'
```

The top-3 views of this query are also ranked in overall top-10 views recommended by View-360, which are shown in Figure 5.12, 5.13, and 5.15. As mentioned in Section 5.5.2, the dataset consists of patients that have different primary diagnosis. For simplification the primary diagnosis `diag_1` attribute has been divided into five broad categories, which is shown on the x-axis as the dimension attribute in Figure 5.12, 5.13, and 5.15.

Particularly, in Figure 5.12 the comparison view shows the probability distribution of `COUNT` of all patients for defined categories of primary diagnosis attribute (`diag_1`). Figure shows that 30% of the patients have circulatory diseases as primary diagnosis. While less than 10% have diabetes and another 10% have injury related primary diagnosis. The target view in Figure 5.12 shows the probability distribution of `COUNT` of all patients in age group `'[10,20)'` for categories of primary diagnosis dimension. The distribution shows that 70% of this age group patients have primary diagnosis of diabetes. As mentioned before this dataset is based on the patients for whom any kind of diabetes was entered to the system as a diagnosis. However, the primary diagnosis is the main disease for which the patients are treated. This is a unique insight that patients of age group `'[10-20)'` are primarily diagnosed with diabetes. Despite of the fact that this view is recommended by View-360 after it passed

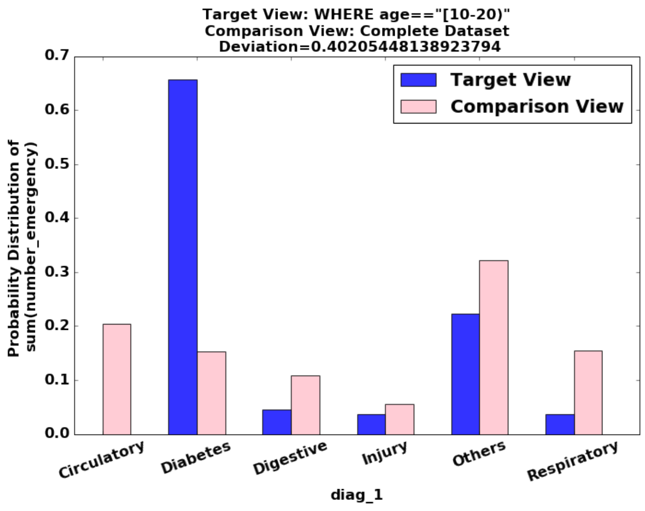


Figure 5.13: Diabetes Dataset: Age Group `[10-20)` V_2

the statistical significance and power test, View-360 provides the option to explore the frequency distribution of a particular attribute. The frequency distribution is shown in Figure 5.11h that this age group has a reasonable size to have an effect.

Figure 5.13 shows the probability distribution of the `SUM` of `Number of emergency` visits in the year preceding the encounter on the y-axis and primary diagnosis `diag_1` on the x-axis. The trend shown in the target view is the same as Figure 5.12 i.e., the 70% of the `SUM` of `Number of emergency` visits in the year preceding the encounter were from the patients diagnosed with diabetes as primary diagnosis. To make sure that normalization has not introduced any bias, the further analysis feature of View-360 is used and the bar-charts of the attribute `diag_1` are viewed. In Figure 5.14 bar charts corresponding to comparison and target views of V_1, V_2, and V_3 are shown. Particularly, Figure 5.14d & 5.14e show the target views of Figure 5.12 and Figure 5.13 respectively without normalization and it confirms the same trend. The comparison view in Figure 5.13 shows that the `SUM` of `Number`

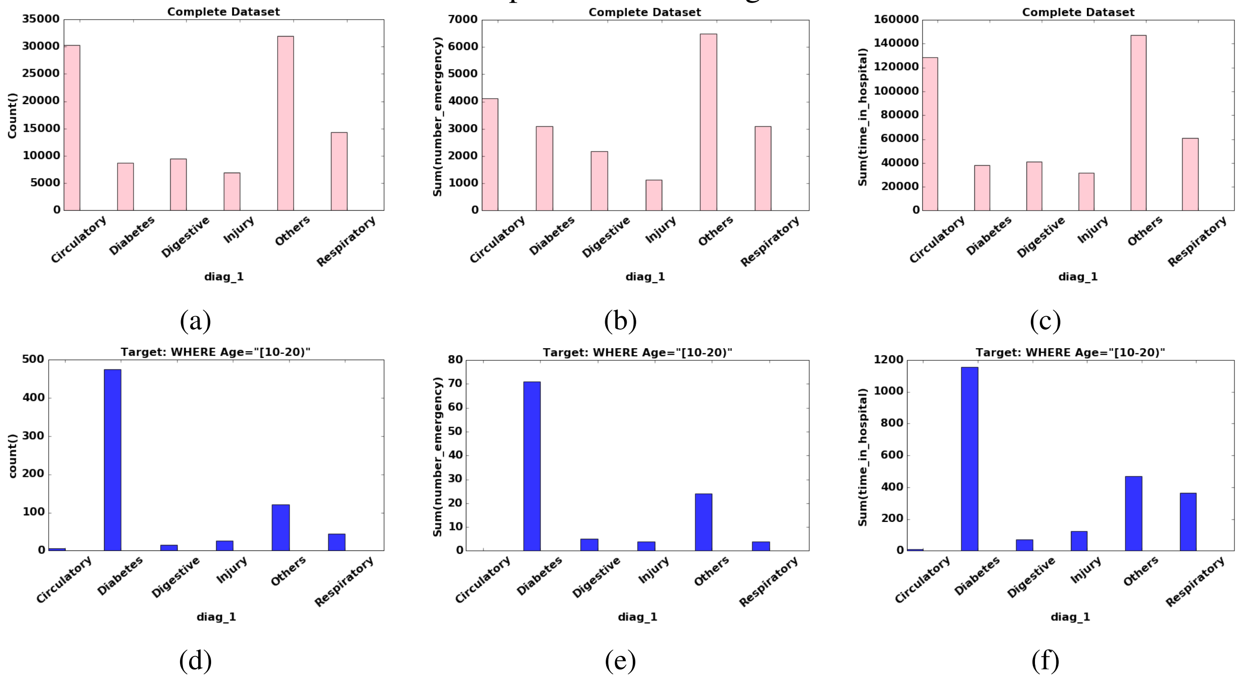


Figure 5.14: Diabetes Dataset: Primary Diagnosis Details

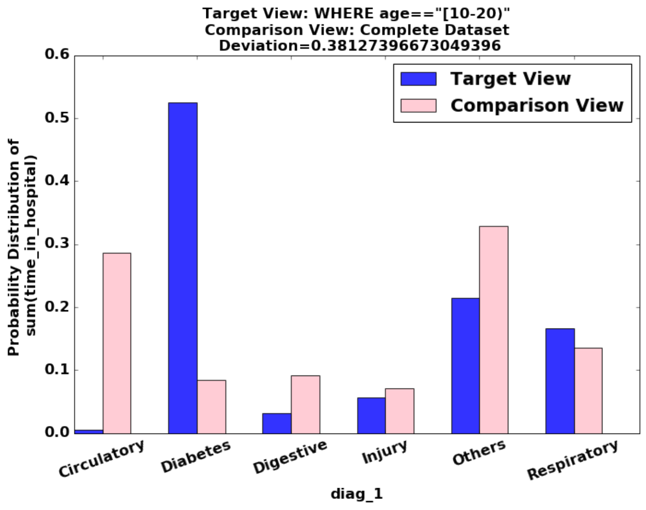


Figure 5.15: Diabetes Dataset: Age Group `[10-20)` V_3

`of emergency` visits in the year preceding the encounter for patients with circulatory disease is 20% and patients with diabetes is 18%. However, from Figure 5.12, we know that the `COUNT` of these patients was 30% and 8% respectively. This can imply firstly, that we have around 10% patience with circulatory diagnosis having no previous emergency visits. Secondly, there are around 8% patients with diabetes as primary diagnosis but they have 18% of the sum of emergency visits in preceding year. Figure 5.14a & 5.14b also confirm these insights.

Figure 5.15 shows the probability distribution of the `SUM` of `Time in hospital` (in terms of number of days between admission and discharge) on the y-axis and primary diagnosis `diag_1` on the x-axis. For comparison view it shows exactly same pattern as Figure 5.12, which means generally no unique observation can be made about the days spend in the hospital. However, for the target view i.e., patients of age group '`[10-20)`', there are some key observations. Firstly, the sum of the time spent in hospital for patients with diabetes as primary diagnosis is around 50% while from Figure 5.12, we know that these patients are around 70% in number. This implies diabetic patients are admitted for less amount of time in hospital as compared to patients diagnosed with other diseases for this age group. Secondly, patients with respiratory disease as primary diagnosis has around 20% of the sum of time in hospital, while these patients are around 5% in number as can be seen in Figure 5.12. This implies this age group patients with respiratory disease as primary diagnosis are admitted in the hospital for longer period of time. This can be seen from Figure 5.14d & 5.14f as well.

Analysis 2: Diabetes Types

For this analysis, View-360 is set to consider the diabetes patients data only. The top visualization recommended by View-360, offered insights into the type of diabetes diagnosis and its relation with emergency visits recorded in the preceding year. The type of diabetes is encoded in the attribute `diag_1` (primary diagnosis), Table 5.4 shows the details of the codes.

1. First insight shown in Figure 5.16, is about those diagnosed with diabetes type 250.0x (i.e., `diag_1=='2500'`). Figure 5.16 shows the `A1C results` (i.e., Hemoglobin A1c abbreviated as HbA1c blood test result) on the x-axis and the probability distribution of the `MEAN` of `Number`

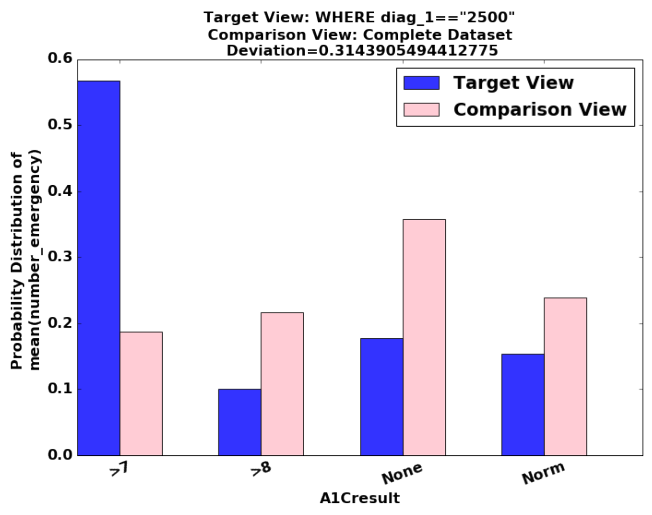


Figure 5.16: Diabetes Type 250.0x vs. HbA1CResult

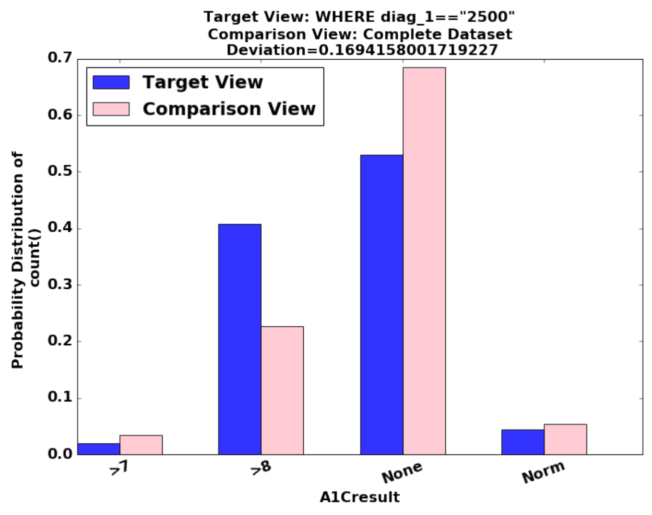


Figure 5.17: Diabetes Type 250.0x vs. HbA1CResult

`of emergency` visits in the preceding year of encounter on the y-axis. The comparison view is for all diabetic patients and it shows in the figure that the highest percentage of the `MEAN` of `Number of emergency` visits in preceding year is for those patients who were not tested for HbA1c and the lowest is for those patients who have their test value > 7. This implies that the patient having any type of diabetes and are not tested for HbA1c test have been admitted more often in emergency in previous year. The Target view shows the probability distribution of the patients who have been diagnosed with diabetes type 250.0x. Among these patients around 55% of the mean of emergency visits is for those who have Hb1Ac test result > 7. This means if a patient is diagnosed with 250.0x diabetes and HbA1C test result is > 7 then there are more chances of coming to emergency again. To confirm this insight, further analysis is performed in View-360 by exploring the the corresponding aggregate view with `COUNT` as an aggregate function, as shown in Figure 5.17. Note that this view has a lower rank that is why it did not come in the top-k automatically. This views shows that for target view the percentage of patients with test value > 7 is less than 5%, but in Figure 5.16 their percentage is highest, that means they are admitted in emergency number of times in the preceding year.

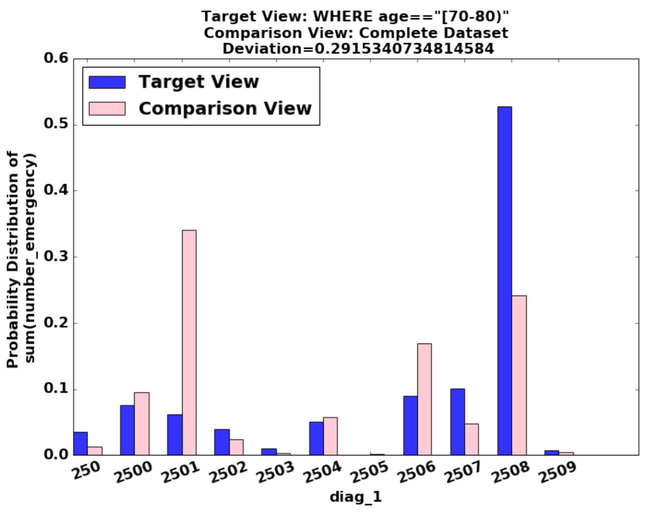


Figure 5.18: Diabetes Types and Age group 70-80

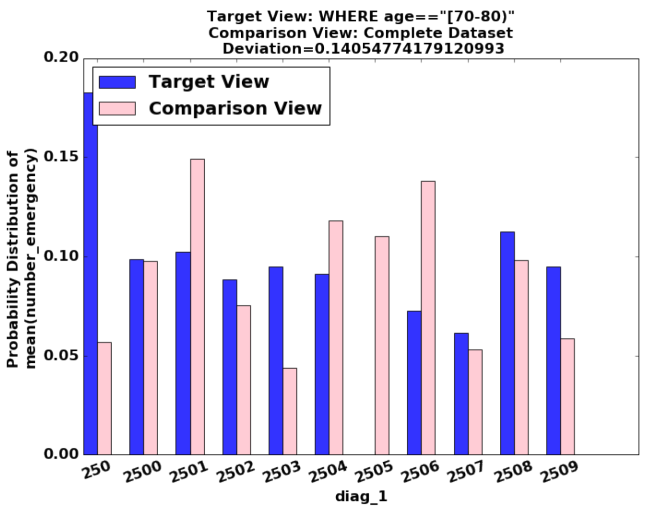


Figure 5.19: Diabetes Types and Age group 70-80

2. The view shown in Figure 5.18 is related to diabetes types and age group 70-80. In the figure the target view is from refined query `WHERE Age=='[70-80)'` which is generated by employing QuRVe in View-360. The Figure shows primary diagnosis attribute `diag_1` on the x-axis and the `SUM` of `Number of emergency` visits in preceding year on the y-axis. The comparison view on all the patients shows that 35% the sum of emergency visits is for the diabetes type 250.1x. This means either there are comparatively more number of patients diagnosed with 250.1x diabetes type or the patients who have been diagnosed with this type tend to have more emergency visits. The target view shows that more than 50% of the sum of emergency visits for patients within the age group 70-80 is for diabetes type 250.8x. As the aggregate function here is `SUM` therefore, further investigation is required. In Figure 5.19, the view with same dimension, measure and predicates but `MEAN` as aggregate function is shown, which is generated by the further exploration feature of View-360. It can be clearly seen that the distribution of target and comparison views are quite similar, that mean the Figure 5.18 is not a true insight.

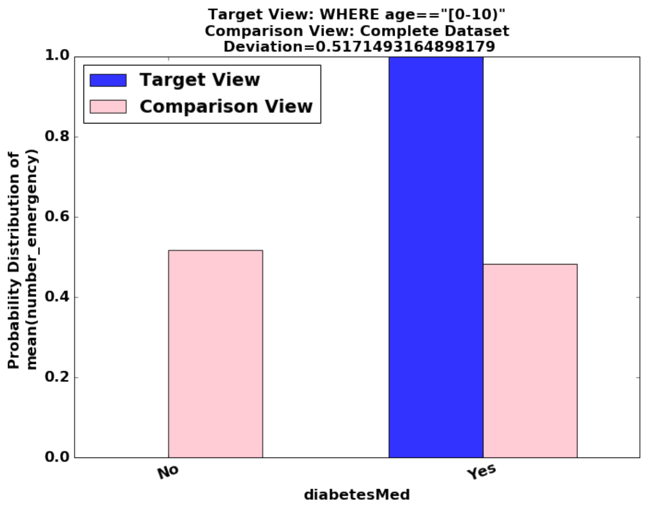


Figure 5.20: Diabetes Dataset: Age Group [0-10) V_1

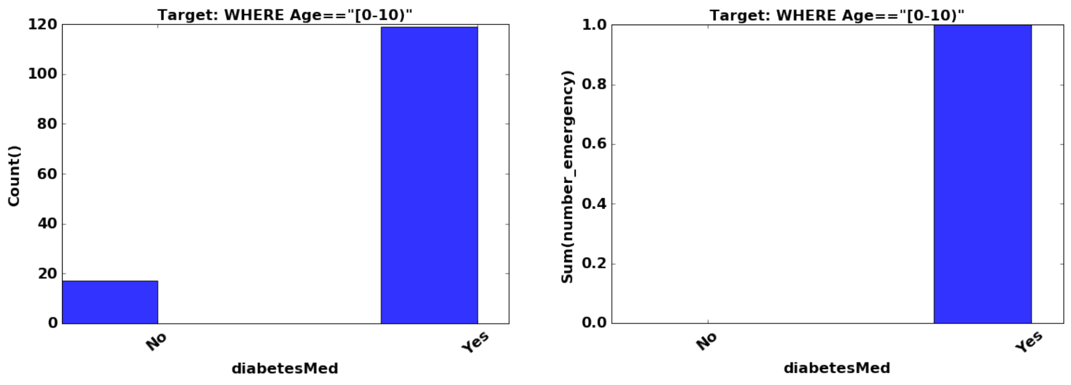
Analysis 3: Age group '[0-10)' years

This analysis is based on one of the highly rated refined query by View-360. Specifically, it is related to those diagnosed with diabetes and are placed in age group 0 to 10 years, which means it is corresponding to the following refined query.

```
SELECT * FROM Diabetes-Subset WHERE Age=='[0-10)'
```

First refer back to Figure 5.11b, where it can be seen that this age group is the second smallest age group present in the dataset. Therefore, this need to be kept in mind that the effect of the views related to this group is small due to small sample size.

The Figure 5.20 shows the attribute `DiabetesMed` (indicating if there was any diabetes medicine prescribed) on the x-axis and the probability distribution of the `MEAN` of `Number of emergency` visits in preceding year on the y-axis. The comparison views shows that for all patients the `MEAN` of `Number of emergency` visits is equally distributed between patients who are on medicine and patients who are not on diabetes medicine. However, in the target view, which is corresponding to patients of age group '[0-10)', it shows that almost all of emergency visits in preceding year are from patients which are on diabetes medicine. The target view triggered further analysis to understand the details of this insights.



(a) (b)

Figure 5.21: Diabetes Dataset: Age Group [0-10) Bar Charts

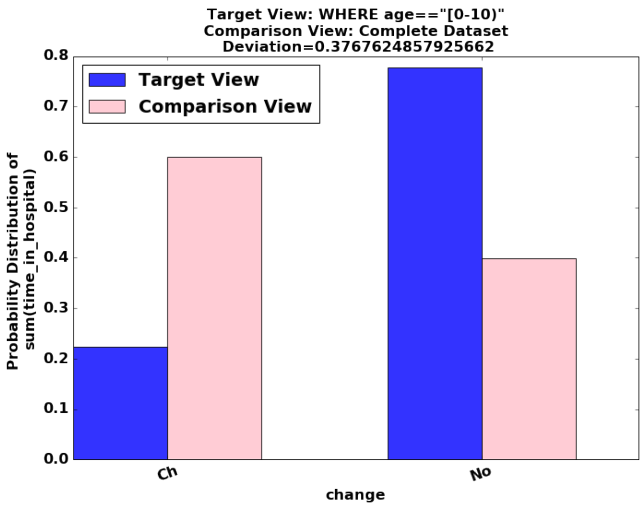


Figure 5.22: Diabetes Dataset: Age Group [0-10) V_2

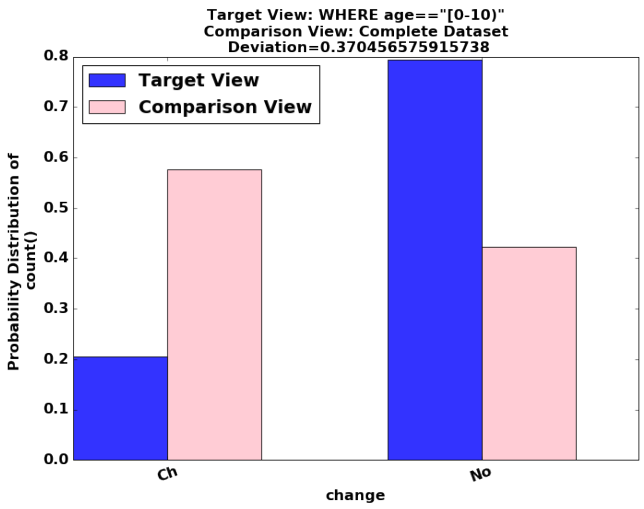


Figure 5.23: Diabetes Dataset: Age Group [0-10) V_3

Therefore, the further exploration feature of View-360 is used and bar charts shown Figure 5.21 are generated. Figure 5.21a shows the number of patients of age group '[0-10)' in each category of `diabetesMed`. It can be clearly seen that there exists tuples in both categories. Then View-360 further shows another bar chart as shown in the Figure 5.21b. In Figure 5.21b, the `SUM` of `Number of emergency` visit attribute revealed that among all the patients of this age group only one emergency visit was recorded in the preceding year. This resulted in the extreme view of the Figure 5.20.

Figure 5.22 shows a view V_2 with the attribute `Change` in medicine on the x-axis and the probability distribution of the `SUM` of the `Time in hospital` on the y-axis. In the comparison view of the figure it can be seen that almost 60% of `SUM` of `Time in hospital` is for patients who had their medicine changed while rest had no change in medicine. For the target view corresponding to the patients of age group '[0-10)' years, 80% of the `SUM` of `Time in hospital` is for patients who had no change in medicine. The next view shown in the Figure 5.23 is exactly the same view with one difference that the aggregate function is `COUNT`. Therefore, it means that the view with `SUM` only recommended by View-360 in top-k due to having the distribution of data of comparison and target view as shown by `COUNT` in Figure 5.23 and not due to a different trend in the hospital stay.

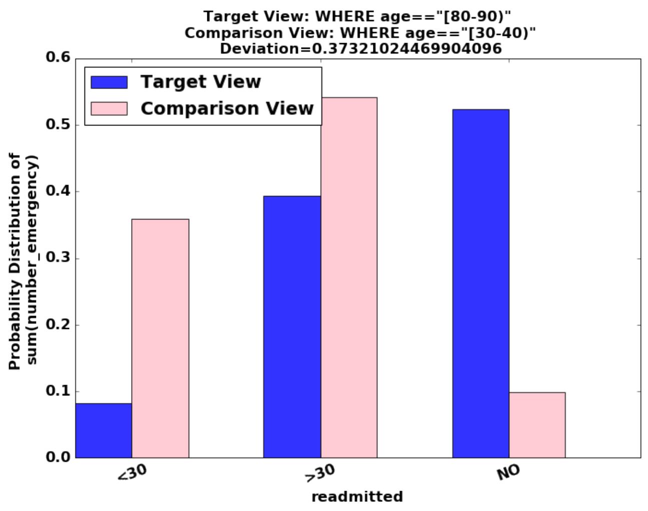


Figure 5.24: Diabetes Dataset: Re-admissions for Different Age Groups V_1

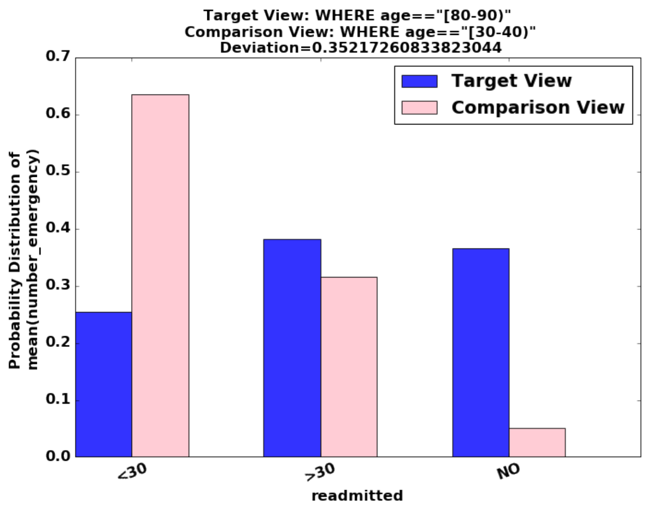


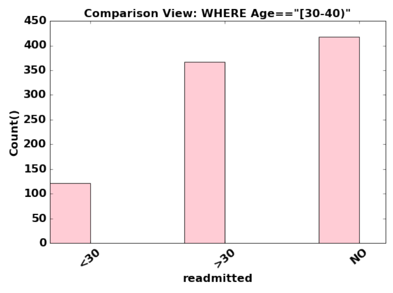
Figure 5.25: Diabetes Dataset: Re-admissions for Different Age Groups V_2

Analysis 4: Re-admissions

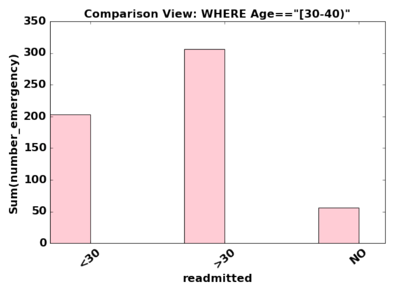
As mentioned in Section 5.5 that in the recent years, the interest in reducing diabetes hospital re-admissions has increased due to its potential to reduce healthcare costs and improve care because of the growth of the burden of diabetes. Therefore, to study the factors related to re-admissions, in View-360 the top-k views are combined with respect to the dimesion attributes and then the views related to `Readmitted` dimension are analyzed.

The views identified some unique insights related to readmission trend of specific groups of patients. This particular analysis is based on views that are generated after refinement on in put query and reference dataset refinement. For the views shown in this analysis attribute `Readmitted` is the dimension attribute which is shown on the x-axis. As mentioned in Table 5.3, the attribute `Readmitted` has three categories; 1) "$<$ 30": if the patient was readmitted in less than 30 days, 2) "$>$ 30": if the patient was readmitted in more than 30 days, and 3) "No": if there is no record of readmission.

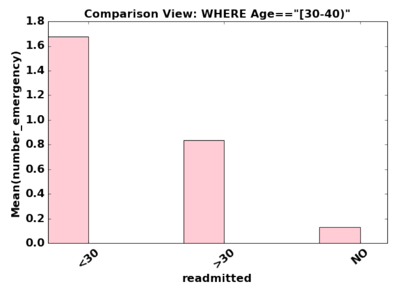
In the aggregate views shown in Figure 5.24 & 5.25 , the target views are from refined query `WHERE Age=='[80-90)'`, while the comparison views are from refined query `WHERE Age=='[30-40)'`. The age groups '`[30-40)`' and '`[80-90)`' are the 5th and 6th largest groups in the population of 10



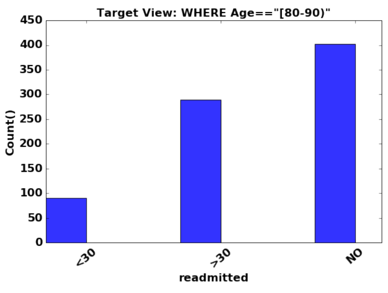
(a)



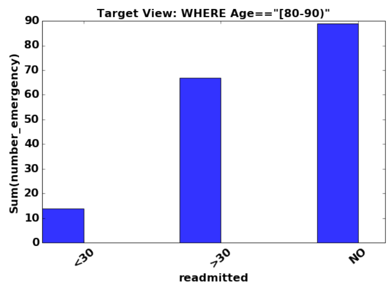
(b)



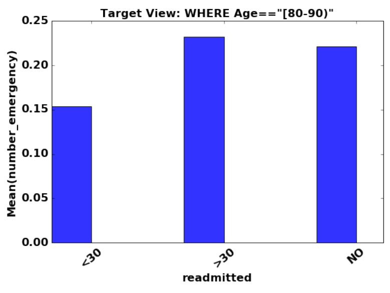
(c)



(d)



(e)



(f)

Figure 5.26: Diabetes Dataset: Re-admissions for Different Age Groups Bar Charts

groups as can be seen in Figure 5.11b. This means the insights drawn from these subsets of data have big effect in terms of statistical significance.

The view in Figure 5.24 shows the probability distribution of the `SUM` of `Number of emergency` visits in preceding year on the y-axis. The comparison view (corresponding to age group '`[30-40)`') shows that the 55% of the `SUM` of `Number of emergency` visits in preceding year are for patients that are readmitted after more than 30 days, while 35% are for patients who are readmitted in less than 30 days. The target view shows that for patients of age group `Age=='[80-90)'`, around 50% of the `SUM` of `Number of emergency` visits in preceding year are for patients who have no record of readmission. This implies that in this age group the patience who had the highest sum of emergency visits in preceding year were not readmitted in this year. However, the views with aggregate function `SUM` are non conclusive as the values can be due to number of factors such as, the distribution of count of the categories, some outliers values in the measure attribute. Therefore, using View-360's further exploration feature, other views for same target and comparison subsets, same dimension and measure attribute, but different aggregate function are considered. Figure 5.25 shows a view with i.e. `MEAN` as aggregate function and ranked close to the view of Figure 5.24. This mean the pattern shown in Figure 5.24 is semantically significant. The Figure 5.25 shows a completely different pattern than Figure 5.24, therefore further exloration is triggred through View-360. The individual bar charts with `COUNT, SUM, MEAN` aggregate functions are generated, `Number of emergency` visits as measure, for target and comparison views as shown in Figure 5.26a- 5.26f.

Figure 5.26a and Figure 5.26d shows similar distribution of target and comparison age groups of patients for all categories of `readmitted`, that is why the aggregate view with `COUNT` was ranked low in the list of top-k. However following key observation can be made from the other views.

1. In Figure 5.26b, the comparison view (corresponding to age group '`[30-40)`') shows that

the highest SUM of Number of emergency visits in preceding year are for patients that are readmitted after more than 30 days. This is probably because these patients are higher in number as shown in Figure 5.26a and not because these patients were getting sick again and again in the preceding year.

2. In Figure 5.26b, the lowest SUM of Number of emergency visits in preceding year are for patients that have no recorded readmission, however, these patients are highest in number as shown in Figure 5.26a. This means this is the healthiest category of patients, who had non-existent emergency encounters in preceding year and in this encounter they did not need a readmission. Figure 5.26c confirms this insight as these patients have the smallest MEAN of Number of emergency visits.

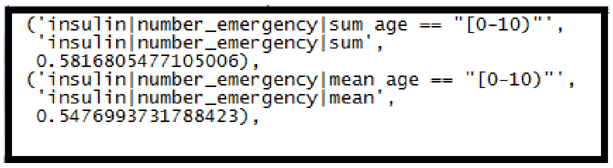
3. In Figure 5.26b, around 40% of the SUM of Number of emergency visits in preceding year is for patients that are readmitted in less than 30 days, although these patients are smallest in number as shown in Figure 5.26a. This implies the patients who are readmitted in less than 30 days, had higher number of emergency visits in the previous year as well. Figure 5.26c shows that these patients have the highest MEAN of Number of emergency visits. In short the patients of age group '[30-40)', having high number of emergency visits in preceding year, have high probability of readmission in less than 30 days.

4. In Figure 5.26e, the target view (corresponding to age group '[80-90)') shows that the highest SUM of Number of emergency visits in preceding year are for patients that are not readmitted. This is contrary to expected pattern, which is if the patients had more emergency visits in last year, they are expected to have readmission, as was shown in the comparison view analysis.

5.6 Discussion

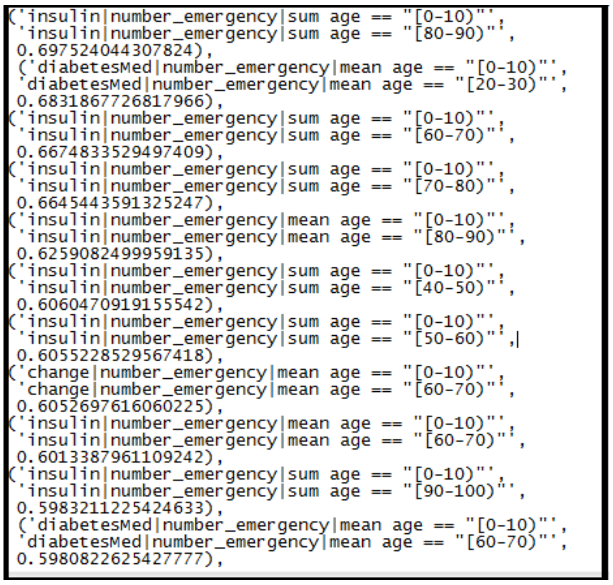
The detailed analysis of two datasets in the previous sections highlight the effectiveness of proposed techniques for view recommendation. Particularly, we show that the interestingness of the recommended views improve in terms of deviation by automatically generating best binning on numerical attributes by employing MuVE, automatically finding subsets of data by employing QuRVe which automatically refines input query and finally automatically findings two interesting subsets of data by refining reference dataset. The View-360 seamlessly performs all of these tasks and facilitate the user in the exploration process. However, by no means we claim that we have fixed all open issues in this problem domain. Rather, we believe that the View-360 is just an initial step towards having a holistic system that effectively recommends interesting views for data exploration. In this section we discuss the lessons learnt from View-360 and some of the open questions related to it.

Attribute Sets: One of the key factors on which interestingness of a view depends is the attributes used for measure, dimension and predicate. Although, View-360 searches for interesting views from all combinations of A,M, and P, however, the user provides us with the sets $\mathbb{A}$,$\mathbb{M}$ and $\mathbb{P}$ as input. Identifying meaning and relative importance of attributes is a non-trivial task and depends completely



```
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum',
 0.5816805477105006),
('insulin|number_emergency|mean age == "[0-10)"',
 'insulin|number_emergency|mean',
 0.5476993731788423),
```

Figure 5.27: Top-k List with Target View Refinement



```
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[80-90)"',
 0.697524044307824),
 ('diabetesMed|number_emergency|mean age == "[0-10)"',
 'diabetesMed|number_emergency|mean age == "[20-30)"',
 0.6831867726817966),
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[60-70)"',
 0.6674833529497409),
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[70-80)"',
 0.6645443591325247),
('insulin|number_emergency|mean age == "[0-10)"',
 'insulin|number_emergency|mean age == "[80-90)"',
 0.6259082499959135),
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[40-50)"',
 0.6060470919155542),
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[50-60)"',|
 0.6055228529567418),
('change|number_emergency|mean age == "[0-10)"',
 'change|number_emergency|mean age == "[60-70)"',
 0.6052697616060225),
('insulin|number_emergency|mean age == "[0-10)"',
 'insulin|number_emergency|mean age == "[60-70)"',
 0.6013387961109242),
('insulin|number_emergency|sum age == "[0-10)"',
 'insulin|number_emergency|sum age == "[90-100)"',
 0.5983211225424633),
 ('diabetesMed|number_emergency|mean age == "[0-10)"',
 'diabetesMed|number_emergency|mean age == "[60-70)"',
 0.5980822625427777),
```

Figure 5.28: Top-k List with Comparison View Refinement

on the semantics of the data. This is an open question that how to decide which attribute is relevant to which set. One straight forward strategy is that all dependent and numeric attributes can be assigned to set of measures ($\mathbb{M}$), while All independent and categorical attributes can be assigned as dimensions ($\mathbb{A}$). However, some attributes are suitable for both predicates and dimensions. In View-360 the choice is left for the user and if the user defines overlapping sets of $\mathbb{A}$ and $\mathbb{P}$, View-360 assigns the overlapping attributes one role at a time, i.e., if the attribute is used as a dimensions it is removed from predicates and vice versa.

Aggregate Functions: View-360 supports `COUNT, SUM, AVG` as aggregate functions in our analysis. When the list of top-k aggregate views is generated each view is ranked as an individual, however, in the analysis, it was noticed that just one particular view with a particular aggregate function fails to tell a complete story about the data. For instance, in Figure 5.16, just one view was not enough to tell the whole picture, views with other aggregate functions were displayed in View-360's further exploration feature, as shown in Figure 5.17, to complete the picture. In short, to understand the insight, all the views with all aggregate functions are considered, therefore, it might be a good idea to group together the views with the same predicates, dimensions and measures but different aggregate functions and then rank the groups to get the insights.

Quality of Views: When View-360 was configured to refine the reference dataset, a big boost in the deviation of the recommended views was observed, but it compromised semantic quality of the recommended views. Simply searching for two subsets that are completely different from each other on some combination of A,M and F can be extremely noisy and misleading. Automatic refinement results in very restrictive queries that represent a small subset of data, the power analysis of View-360 checks that the subset passes the minimum criteria, however it appears that this is not enough to make sure interesting insight. The smallest subset that passes the power test comes out in top-k with every other subset. For instance, Section 5.5.1 after automatic refinement the smallest subset comes out to be patients with age group '[0-10)' and the top-2 views belong to that subset as shown in Figure 5.27. While Figure 5.28 shows list of top-k views, when the refinement is applied to reference dataset as well, it can be clearly seen that age group '[0-10)' is compared with every other age group that exist in data. These views have high deviation but not really interesting semantically. Moreover, such views provide little information gain and are less interesting for the user. However, how to detect these views and prune them is challenging and is an interesting direction for the future work.

Ranking Criteria: View-360 ranks the views based on the deviation between target and comparison views, however, the interestingness can be explored by incorporating other criterion. For instance, in Figure 5.26b, the comparison view in itself was showing a unique pattern when compared with the comparison view of Figure 5.26a instead of the target view. This mean in one aggregate view if the corresponding target and comparison view are based on different aggregate function instead of different predicates it can bring out something new and interesting. Moreover, the conversion of results into probability distribution is useful generally, however, in some cases it leads to misleading results and investigating without normalization gives better insights into data. On similar lines other ranking criterion and possibilities need to be explored.

User Feedback: View-360 gives weight to the user's preferences by allowing the user to specify number of input parameters and then making the recommendations based on automatic exploration. View-360 provides maximum coverage by exploring all possible subsets of data and making all possible comparisons. However, exploration is an iterative process, it is impossible to guarantee that the recommended views satisfy user's expectations. In most cases, the user do explore in iterations by changing input parameters. Despite of all the automation, user still is the key to effective recommendation of views. Therefore, it is worth investigating how to improve quality of recommended view according to feedback from the user. Additionally, history of exploration from same user or other users on same dataset can also be used as an input to the recommendation process.

Chapter 6

Conclusions and Future Work

The goal of this thesis was the design, implementation and evaluation of view recommendation schemes for visual data exploration. Next, in Section 6.1, we summarize our contributions towards that goal and in Section 6.2, we describe directions for future work.

6.1 Summary of Contributions

We have addressed the challenging problem of efficiently and effectively recommending views from complex datasets for visual data exploration. While the recommended views provides the user with effortless insights into data, quantification of relevance for view recommendation is a non-trivial task and, additionally, the recommendation process is tremendously computationally expensive. Hence, in order to address these challenges we proposed various schemes in this thesis as summarized below.

In Chapter 3, we proposed a novel utility function and a suite of search schemes for recommending top-k views in the presence of numerical dimensions. Our utility function recognizes the impact of numerical dimensions on visualization, which is captured by means of multiple objectives, namely: deviation, accuracy, and usability. Our proposed search schemes further incorporate that utility function for the purpose of recommending the top-k aggregate data visualizations. A key goal in the design of those search schemes is to efficiently prune the prohibitively large search space of possible views. That goal is reasonably achieved by our first scheme *Multi-Objective View Recommendation for Data Exploration (MuVE)*, and is further improved by *uMuVE*, at the expense of a high memory usage. Accordingly, we presented *MuMuVE* , which provides a pruning power close to that of uMuVE, while keeping memory usage within a predefined constraint. Our extensive experimental results show the significant gains provided by our proposed scheme.

The most expensive operation while computing the utility of the views is the time spent in executing the query related to the views. To reduce the cost of this particular operation, in Chapter 3.5, we propose a novel technique, *materialized View (mView)*, which instead of answering each query related to a view from scratch, reuses results of the already executed queries. This is done by incremental materialization of a set of views in optimal order and answering the queries from the materialized views instead of the base table.

Visual data exploration involves several iterations of selecting subset of data by issuing an input query, and analysis by generating different visualizations. Motivated by the need for finding interesting views from prudent subsets of data (i.e., input queries), in Chapter 4, we propose efficient schemes *Query Refinement for View Recommendation (QuRVe)*, that automatically refine input query to search for subsets of data having interesting views and recommend the top-k views. However, such uncontrolled refinement of queries can lead to multiple problems such as loss of user preference and random discoveries. Therefore, a multi-objective function is proposed to measure relevance, interestingness and significance of the refined queries and their corresponding views. We have proposed a novel suit of schemes that efficiently navigate the refined queries search space for recommendation of data visualizations. The main idea underlying the proposed QuRVe scheme is to incrementally access the refined queries in order of their similarity with the original query, which allows an early termination of search and results in pruning of a large number of views. Additionally, *uQuRVe* scheme reduces the cost further by tightening the upper bounds on the utility of the views and short circuiting unnecessary views. In addition, *uQuRVe-range* scheme is proposed, which makes sure that high utility views are probed first and as a result higher number of low utility views are pruned. We have also proposed approximation based schemes that provide order of magnitude reductions in processing costs, while maintaining utility of recommended views near optimal schemes. Our extensive experimental evaluations show the efficiency exhibited by our proposed schemes under various settings, and the the significant benefit it provides compared to existing methods.

QuRVe focused on finding interesting views from all subsets of data by comparing them with a user provided reference dataset. However, the search space can be further extended by involving comparison of all subsets of data with each other to find interesting views. Therefore, to explore this dimension of the problem, in Chapter 5 we propose to automatically refine query for comparison views as well. We propose the context of comparison between the target and comparison view queries. We also outline the design and implementation of a holistic prototype system *View-360*, which included all aspects of aggregate view recommendation i.e, recommendation based on categorical and numerical attributes, and recommendation based on refinement on target and comparison queries. We then showcase the effectiveness of our proposed schemes by performing detailed analysis on two real datasets from different domains.

6.2 Future Work

In this section, we propose possible directions for the future work.

Interactivity

In the future, the data-driven approach adopted in this work can be extended to incorporate a user-driven approach for recommending data visualizations to achieve the right balance between interactivity and automation. The balance between automation and interactivity is the key for effective analysis, as we saw in Chapter 5, merely providing the user top-k ranked views falls short in terms of revealing immediate insights. Therefore, in the future, we propose to provide the user with more sophisticated interface to investigate the recommended views further, in iterations. Consequently, this task can be aided with automatic explanations (textual and visual) and history of exploration, to help the user understand the recommended view.

User Feedback

Although View-360 incorporates user-preference in terms of input parameters. However, in future, we propose to further investigate different methods for capturing different aspects of the user's preference and incorporating it in the recommendation process. For instance, in an interactive approach, the user can be presented with a small set of sample views and they can be requested to provide relevance feedback (i.e., if the view is relevant to their analysis task), similar to the approach proposed in [29]. That feedback can then be used to build a predictive model to learn the user's preference, which can be integrated with our data-driven model. Moreover, we can leverage user feedback to learn the type of views that a user finds interesting and use that model to prune uninteresting views. Orthogonally, to extend our problem for the cases where recommendations can be made based on the availability of a history of views that the user has found to be interesting in the past, or that have been identified as interesting by similar users. As future work, we can investigate the integration of collaborative filtering model with our approach.

Broad Insight Space

We measure interestingness as the distance (deviation) between a target view and a comparison view. However, in general, one size does not fit all principle applies here. The interestingness can have different definitions for different users, for instance, it can be a strong manifestation of a distributional property of the data, such as skewness, strong correlation, tight clustering, or it can be unexpectedness of a pattern. Therefore, in future, we plan to integrate other measures of interestingness to have more general audience for View-360.

Secondly, from our extended analysis of datasets in Chapter 5, we have built a case that an insight can not be just one visualization. It should be a combination of visualizations that tells a story about the data. It needs more work to analyze how to quantify an insight. Particularly, score visualizations such that the ones that are related and build a story are grouped together in one insight and that insight gets scored accordingly.

Bibliography

[1] H. Ehsan et al. Muve: Efficient multi-objective view recommendation for visual data exploration. In *ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 731–742, 2016.

[2] H. Ehsan et al. Efficient recommendation of aggregate data visualizations. *IEEE Trans. Knowl. Data Eng.*, 30(2):263–277, 2018.

[3] H. Ehsan and M. A. Sharaf. Materialized view selection for aggregate view recommendation. In L. Chang, J. Gan, and X. Cao, editors, *Databases Theory and Applications*, pages 104–118, Cham, 2019. Springer International Publishing.

[4] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore. Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

[5] `https://www.ncbi.nlm.nih.gov/books/NBK368403/table/sb203.t5/`.

[6] M. Vartak et al. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[7] A. Key et al. Vizdeck: self-organizing dashboards for visual analytics. In *SIGMOD*, pages 681–684, 2012.

[8] S. Kandel et al. Profiler: integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554, 2012.

[9] T. Sellam et al. Ziggy: Characterizing query results for data explorers. *PVLDB*, 9(13):1473–1476, 2016.

[10] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 101–112, 2018.

[11] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence to sequence recurrent neural networks. *CoRR*, abs/1804.03126, 2018.

[12] K. Z. Hu, M. A. Bakker, S. Li, T. Kraska, and C. A. Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 128, 2019.

[13] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

[14] T. Sellam et al. Fast, explainable view detection to characterize exploration queries. In *SSDBM 2016, Budapest, Hungary, July 18-20, 2016*, pages 20:1–20:12, 2016.

[15] C. Wang and K. Chakrabarti. Efficient attribute recommendation with probabilistic guarantee. In *KDD*, pages 2387–2396, 2018.

[16] D. Gotz and Z. Wen. Behavior-driven visualization recommendation. In *Proceedings of the 14th International Conference on Intelligent User Interfaces, IUI 2009, Sanibel Island, Florida, USA, February 8-11, 2009*, pages 315–324, 2009.

[17] M. Vartak et al. SEEDB: automatically generating query visualizations. *PVLDB*, 7(13):1581–1584, 2014.

[18] G. Cormode et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[19] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[20] `https://archive.ics.uci.edu/ml/datasets/adult`.

[21] S. Idreos et al. Overview of data exploration techniques. In *SIGMOD*, pages 277–281, 2015.

[22] A. Giuzio, G. Mecca, E. Quintarelli, M. Roveri, D. Santoro, and L. Tanca. INDIANA: an interactive system for assisting database exploration. *Inf. Syst.*, 83:40–56, 2019.

[23] P. Kubernátová, M. Friedjungová, and M. van Duijn. Constructing a data visualization recommender system. In *International Conference on Data Management Technologies and Applications*, pages 1–25. Springer, 2018.

[24] I. Dankwa-Mullan, M. Rivo, M. Sepulveda, Y. Park, J. Snowdon, and K. Rhee. Transforming diabetes care through artificial intelligence: The future is here. *Population Health Management*, 22(3):229–242, 2019. PMID: 30256722.

[25] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 49–60, 2013.

[26] Q. T. Tran, C. Chan, and S. Parthasarathy. Query by output. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 535–548, 2009.

[27] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: an automatic query steering framework for interactive data exploration. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 517–528, 2014.

[28] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 493–504, 2014.

[29] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. AIDE: an active learning-based approach for interactive data exploration. *IEEE Trans. Knowl. Data Eng.*, 28(11):2842–2856, 2016.

[30] M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *VLDB J.*, 22(6):849–874, 2013.

[31] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1262–1265, 2014.

[32] A. Nandi. Querying without keyboards. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[33] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Playful query specification with dataplay. *PVLDB*, 5(12):1938–1941, 2012.

[34] A. Kalinin, U. Çetintemel, and S. B. Zdonik. Interactive data exploration using semantic windows. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 505–516, 2014.

[35] L. Jiang and A. Nandi. Snaptoquery: Providing interactive feedback during exploratory query specification. *PVLDB*, 8(11):1250–1261, 2015.

[36] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware auto-completion for SQL. *PVLDB*, 4(1):22–33, 2010.

[37] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. In *CIDR*, 2013.

[38] P. Neophytou, R. Gheorghiu, R. Hachey, T. Luciani, D. Bao, A. Labrinidis, G. E. Marai, and P. K. Chrysanthis. Astroshelf: understanding the universe through scalable navigation of a galaxy of annotations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 713–716, 2012.

[39] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. G. Parameswaran. Towards visualization recommendation systems. *SIGMOD Record*, 45(4):34–39, 2016.

[40] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence to sequence recurrent neural networks. *IEEE computer graphics and applications*, 2019.

[41] `public.tableau.com`.

[42] `www.qlik.com`.

[43] C. Ahlberg. Spotfire: An information exploration environment. *SIGMOD Record*, 25(4):25–29, 1996.

[44] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati. Foresight: Recommending visual insights. *PVLDB*, 10(12):1937–1940, 2017.

[45] J. Seo et al. Knowledge discovery in high-dimensional data: Case studies and a user survey for the rank-by-feature framework. *IEEE Trans. Vis. Comput. Graph.*, 12(3):311–322, 2006.

[46] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1363–1375, 2016.

[47] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.*, pages 317–332, 2019.

[48] L. D. Stefani, L. F. Spiegelberg, T. Kraska, and E. Upfal. Vizrec: A framework for secure data exploration via visual representation. *CoRR*, abs/1811.00602, 2018.

[49] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1509–1524, 2017.

[50] E. Zgraggen, Z. Zhao, R. C. Zeleznik, and T. Kraska. Investigating the effect of the multiple comparisons problem in visual analysis. In *CHI*, page 479, 2018.

[51] D. J. L. Lee, H. Dev, H. Hu, H. Elmeleegy, and A. G. Parameswaran. Avoiding drill-down fallacies with *VisPilot*: assisted exploration of data subsets. In *Proceedings of the 24th International Conference on Intelligent User Interfaces, IUI 2019, Marina del Ray, CA, USA, March 17-20, 2019*, pages 186–196, 2019.

[52] Y. Mizuno, Y. Sasaki, and M. Onizuka. Efficient data slice search for exceptional view detection. In *Proceedings of the Workshops of the EDBT/ICDT 2017 Joint Conference (EDBT/ICDT 2017), Venice, Italy, March 21-24, 2017.*, 2017.

[53] T. Matsumoto, Y. Sasaki, and M. Onizuka. Data slice search for local outlier view detection: A case study in fashion EC. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[54] C. Binnig, L. D. Stefani, T. Kraska, E. Upfal, E. Zgraggen, and Z. Zhao. Toward sustainable insights, or why polygamy is bad for you. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[55] Z. Zhao et al. Controlling false discoveries during interactive data exploration. In *SIGMOD*, 2017.

[56] Q. Cui et al. Measuring data abstraction quality in multiresolution visualizations. *IEEE Trans. Vis. Comput. Graph.*, 12(5):709–716, 2006.

[57] E. Bertini. Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2203–2212, 2011.

[58] H. V. Jagadish et al. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[59] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *ICDE*, pages 190–200, 1995.

[60] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[61] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *VLDB*, pages 318–329, 1996.

[62] H. Gupta and I. S. Mumick. Selection of views to materialize in a data warehouse. *IEEE Trans. Knowl. Data Eng.*, 17(1):24–43, 2005.

[63] J. D. Mackinlay et al. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1137–1144, 2007.

[64] C. Stolte et al. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[65] S. Subramaniam et al. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.

[66] Z. Liu et al. *imMens*: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[67] G. Cormode and M. Garofalakis. Histograms and wavelets on probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1142–1157, 2010.

[68] G. Cormode et al. Histograms and wavelets on probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 22(8):1142–1157, 2010.

[69] N. Bruno et al. Generating queries with cardinality constraints for dbms testing. *IEEE Trans. Knowl. Data Eng.*, 18(12):1721–1725, 2006.

[70] A. Marian et al. Evaluating top-*k* queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[71] R. Fagin et al. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[72] I. F. Ilyas et al. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[73] N. Bruno et al. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 369–380, 2002.

[74] H. A. Khan, M. A. Sharaf, and A. Albarrak. Divide: efficient diversification for interactive data exploration. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014*, pages 15:1–15:12, 2014.

[75] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. BRAID: stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[76] `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`.

[77] `www.basketball-reference.com`.

[78] `https://www.kaggle.com/uciml/adult-census-income`.

[79] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *VLDB*, pages 156–165, 1997.

[80] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

[81] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE*, pages 1081–1092, 2013.

[82] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, pages 862–873, 2009.

[83] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[84] A. Albarrak, M. A. Sharaf, and X. Zhou. Saqr: An efficient scheme for similarity-aware query refinement. In *DASFAA*, 2014.

[85] M. Vartak, V. Raghavan, E. A. Rundensteiner, and S. Madden. Refinement driven processing of aggregation constrained queries. In *EDBT*, 2016.

[86] G. Liu et al. Towards exploratory hypothesis testing and analysis. In *ICDE*, 2011.

[87] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire. Data polygamy: The many-many relationships among urban spatio-temporal data sets. In *SIGMOD*.

[88] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD Conference*, pages 15–26, 2010.

[89] S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data Min. Knowl. Discov.*, 5(3):213–246, 2001.

[90] Y. Chung et al. Towards quantifying uncertainty in data analysis & exploration. *IEEE Data Eng. Bull.*, 41(3):15–27, 2018.

[91] J. Cohen. Statistical power analysis for the behavioral sciences (revised ed.), 1977.

[92] A. Telang, C. Li, and S. Chakravarthy. One size does not fit all: Toward user- and query-dependent ranking for web databases. *IEEE Trans. Knowl. Data Eng.*, 24(9):1671–1685, 2012.

[93] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, pages 594–605, 2004.

[94] `https://www.kaggle.com/usdot/flight-delays`.

[95] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *CoRR*, abs/cs/0701155, 2007.

[96] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for OLAP discovery-driven analysis. *IJDWM*, 7(2):1–25, 2011.

[97] M. Joglekar, H. Garcia-Molina, and A. G. Parameswaran. Smart drill-down: A new data exploration operator. *PVLDB*, 8(12):1928–1931, 2015.

[98] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of OLAP data cubes. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, pages 168–182, 1998.

[99] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 93–104, 2000.

[100] W. Raghupathi and V. Raghupathi. Big data analytics in healthcare: promise and potential. *Health Information Science and Systems*, 2(1):3, Feb 2014.

[101] P. Galetsi and K. Katsaliaki. A review of the literature on big data analytics in healthcare. *Journal of the Operational Research Society*, pages 1–19, 07 2019.

[102] B. Shneiderman, C. Plaisant, and B. W. Hesse. Improving healthcare with interactive visualization. *Computer*, 46(5):58–66, May 2013.

[103] `http://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008`.

Appendix A

Multivariable Linear Regression

In multi linear regression, we use a regression model with one dependent and two or more independent variables. In general, the multiple regression equation of Y on X_1, X_2, ..., X_k is given by:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots\dots\dots + b_k x_k \tag{A.1}$$

Let's assume we have two independent variables x_1 and x_2. Then select a set of sample observations from the data. For multivariable regression analysis, first correlations r between all the variables i.e. $r(y,x_1)$, $r(x_1,x_2)$, and $r(y,x_2)$ are computed. Then multiple (combined) correlation R is computed using following formula:

$$R = \sqrt{\frac{(r_{y,x_1})^2 + (r_{y,x_2})^2 - (2r_{y,x_1} r_{y,x_2} r_{x_1,x_2})}{1-(r_{x_1,x_2})^2}} \tag{A.2}$$

If R=0 that means there is no relationship between variables. If R 1 that implies there is strong relationship between variables. Then calculate regression coefficients b with following formulas:

$$b_1 = \frac{r_{y,x_1} - r_{y,x_2} r_{x_1,x_2}}{1-(r_{x_1,x_2})^2} \frac{SD_y}{SD_{x_1}} \tag{A.3}$$

$$b_2 = \frac{r_{y,x_2} - r_{y,x_1} r_{x_1,x_2}}{1-(r_{x_1,x_2})^2} \frac{SD_y}{SD_{x_2}} \tag{A.4}$$

Where SD is the standard deviation.