SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics

Ayushe Gangal
agangal@umass.edu

1. Problem Statement

Data visualization is an integral part of data analysis. Data is visualized to gain better insights from it without reading the whole dataset, as it requires a lot of time to find useful information from raw data manually. These insights may include but are not limited to finding outliers, most essential features, finding correlations, finding factors for further analysis, and even leading to a whole new problem statement. However, manual exploration and visualization of large datasets are neither efficient, nor guarantee optimal performance, as the analysts first need to manually look at all the features, explore relationships between those features, find outliers and analyze them, create all possible visualizations, etc., and then segregate the created visualizations into interesting or not-interesting. Though it has human control, this process can get extraordinarily painstaking and difficult to manage, which may lead to errors and be slow and inefficient.

In the paper "SeeDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics" [1], the authors propose a solution to this problem by proposing a system that automatically identifies and recommends visualizations for visual analysis of datasets. The proposed approach is middleware software that works on top of any dataset and provides the top-k most interesting visualizations for that dataset. The authors have defined the "interestingness" of visualization by a metric called 'utility' which is defined as the deviation between the reference and the target views.

SeeDB works by finding k (a positive integer given as input) aggregate views that have the largest utility values when given a user-specified query on a database, a utility function, and a reference dataset. It determines the utility of visualizations using deviation. Visualizations that show different trends in the query dataset compared to a reference dataset are said to have high utility.

Despite this recommendation-based software, the performance can suffer a lot as,

- The number of visualizations is large, even for a small dataset with a decent number of attributes.
- Calculating the utility for each of these visualizations separately is computationally wasteful as they all have the same underlying data. This wastes time and computational resources.
- There is a trade-off between the interactive time at which the visualizations are generated and the accuracy of the visualizations.

The authors have proposed two methods for optimization to address these issues. These optimization methods are sharing-based optimization and pruning-based optimization.

2. Techniques Implemented

We have implemented the basic execution engine of SeeDB and were successfully able to replicate the results obtained by the authors. To identify the *k* best aggregate views, SeeDB generates a SQL query corresponding to the target and reference view each, which is recursively done for all the aggregate views. In this, the framework serially executes two SQL queries for each possible view. It then computes the distance between the target and the reference view distributions and returns the *k* visualizations with the highest utility.

We have implemented both the proposed optimization methods, namely, sharing-based and pruning-based optimization methods. In order to implement these approaches, we first needed to implement the base model proposed by the authors which included the calculation of the utility metric, the deviation between the reference and the target views, grouped-by Dimension Attributes A, aggregated using Aggregate Functions F on Measure Attributes M.

Each SEEDB visualization could be translated into an aggregate over group-by query on the underlying data. A visualization V_i is represented by a triple *(a, m, f)*. This is called an aggregate view or simply a view. $V_i(D)$ represents the results of grouping the data in D by a, and then aggregating the m values using *f*. $V_i(D_Q)$ represents a similar visualization applied to the data in D_Q.

SEEDB determines the utility of visualizations using deviation. Visualizations that show different trends in the query dataset D_Q compared to a reference dataset D_R are said to have high utility.

View V_i applied to the query data can be expressed as query Q_T below. This is called the target view.

$$Q_T = \text{SELECT a, } f(m) \text{ FROM } D_Q \text{ GROUP BY a} \tag{1}$$

Similarly, View V_i applied to the reference data can be expressed as query Q_R below. This is called the reference view.

$$Q_R = \text{SELECT a, } f(m) \text{ FROM } D_R \text{ GROUP BY a} \tag{2}$$

To ensure that all the aggregate summaries had the same scale, each summary was normalized into a probability distribution (sum = 1). The probability distribution of the target view is denoted as $P[V_i(D_Q)]$ while that for the reference view, is denoted as $P[V_i(D_R)]$.

Given an aggregate view and probability distributions for reference view and target view, we defined the utility of the view as the distance between these two probability distributions. The higher the distance between the two distributions, the more likely the visualization is interesting and therefore higher the utility. If S is the distance function,

$$U(V_i) \ = \ S(P[V_i(D_Q)],\ P[V_i(D_R)]\ ) \tag{3}$$

Various distance functions can be used to compute utility, including Earth Mover's Distance, Euclidean Distance, etc. SEEDB uses the Earth Mover's distance as the default distance function. We used Kullback-Leibler Divergence (K-L divergence) to compute the utility.

Using the terminology defined above, the SEEDB problem can be formally defined as:

Given a user-specified query Q on a database D, a reference dataset D_R, a utility function U and a positive integer k, find k aggregate views $V \equiv (a,\ m,\ f)$ that have the largest values of $U(V)$ among all the views $(a,\ m,\ f)$ while minimizing the total computation time.

The two issues, scale and utility, were addressed by using pruning-based optimization and deviation as a measure for utility. It is proposed that an interesting visualization has a large deviation from a reference.

2.1 Sharing-based Optimization

The sharing optimization of SeeDB minimizes the number of scans of the dataset by intelligently generating queries by merging and creating a batch of these queries, which reduces the number of queries for the dataset. We have Combined Multiple Aggregates, which is writing a single aggregate view query for multiple queries with the same group-by attribute. We have formulated a single query with multiple aggregate functions instead of having a single query per aggregate function.

2.2 Pruning-based Optimization

Pruning-based optimization determines which low-utility view queries to discard as computing them wastes computational resources. We have implemented the Confidence Interval (CI) based pruning, which uses worst-case statistical confidence intervals to bound view utilities, similar to the top-k pruning algorithms. An estimate of the mean utility view V_i and the confidence interval is kept at the end of each phase. A low-utility pruning rule which states that V_i is discarded if the upper bound of the utility V_i is less than the lower bound of the k or more utility views. The system uses the Hoeffding-Serfling inequality to derive worst-case confidence. This is done to discard views with less utility, which is obtained from KL-divergence. Partial results obtained at the end of each phase are used to determine utility (on the basis of the data processed in those phases) and views with low utility are discarded. We have used Hoeffding-Serfling inequality to bound the confidence mean values with number partitions N = 10.

2.3 Dataset

The dataset used for the base implementation of the paper is '**Census Income Data Set',** downloaded from the census dataset [2]. The original dataset contained 15 features and data points 32560. Following are the attributes:

- age: continuous.
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- sex: Female, Male.
- capital-gain: continuous.
- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad Tobago, Peru, Hong, Holland-Netherlands.
- salary-range: <=50K, >50K

2.4 Data Pre-Processing

We applied the following aggregate functions on the dataset to obtain our recommended visualizations: [sum, avg, min, max, count]. The attributes on which we performed the mentioned aggregate functions are: [age, capital_gain, capital_loss, hours_per_week]. We chose these columns as they were of integer data type and thus, it was easier to apply the aggregate functions to them. We chose *'married'* and *'unmarried'* as our target and reference views, respectively.

The following steps were taken to pre-process the census dataset:

- We dropped the column 'fnlwgt' from the relation
- For the column 'marital_status', we combined the column values (shown below) to make two categories: *married* and *unmarried.*
  married : Married_civ_spouse , Married_spouse_absent, Married_AF_spouse, Separated
  unmarried: Divorced, Never_married, Widowed
- We replaced '?' with 'NaN' for the columns like 'occupation,''native_country', and 'workclass.'
- We dropped the rows which contained missing values.

For our base dataset, below are the attributes and functions used:

- dimension_attributes: [workclass, education, relationship, education_num, occupation, race, sex, native_country, salary_range]
- measure_attributes: [age, capital_gain, capital_loss, hours_per_week]
- aggregate_functions: [sum, avg, max, min, count]

2.5 Methodology

Tech-stack and libraries used:

- Python 3.7 as the Programming language
- Jupyter Notebook as the computing platform
- PostgreSQL database
- Matplotlib to generate plots
- Python package psycopg2, which is a Python PostgreSQL Adapter

Once the data is pre-processed, we create multiple relations using the dataset.

- For Shared-based Optimization: We created three new tables, namely: 'all_adults,' 'unmarried_adults', and 'married_adults.'
- For Pruning-based Optimization: We created ten tables since the number of partitions we chose is 10, and all the rows in the original table are split equally (more or less) into these tables.

Post-creation of tables, below are the steps followed to obtain top-K views:

1. *Implementation of Combining Multiple Aggregates using Shared based Optimization:*
   a. Generate all the views using all the combinations of [a,f,m] chosen for the dataset.

      In the case of the Census dataset, we have nine dimensions, four measures, and five aggregate functions which results in 180 possible views. But since we are grouping all aggregates in a single query, we only have to run fewer queries per input table.

b. Execute the group-by queries combining multiple aggregate functions on both target and reference views.

c. Normalize the results obtained from the above executions

  ○ For normalizing the numeric attributes, we converted the query result values to a smaller value by multiplying it with a fraction equivalent to 1/(sum of all the values in the query result)

  ○ While computing the *utility* using KL Divergence, the computations were skewed by NaN/0 values hence we replaced such values with 1e-10.

d. Calculate the utility measures for the obtained query results using the KL-divergence distance function.

e. Sort the obtained views (which contain the [a,f,m] and the corresponding utility measure) in descending order of utility measures.

f. Plot the top-5 (K=5) views out of these.

2. *Implementation of Confidence Interval-Based Pruning based Optimization:*

a. Set the hyperparameters for the Hoeffding-Serfling inequality function (value of delta and number of phases, N = 10) and initialize a view set with all possible views.

b. In each phase, send a chunk of data to get all the aggregate views generated from the view set. For each of these views, calculate the utility score using KL-divergence and create a list of views with their list of utility scores till this phase.

c. From the second phase, calculate the threshold bound of mean utility scores using Hoeffding-Serfling inequality for each of the views in the current view set and sort these values in descending order.

d. Drop all the view records from the list that have the sum of running mean score and epsilon lower than the acceptable lower bound calculated in the previous step.

e. Repeat steps c-e for N times - once per phase with the pruned set of views.

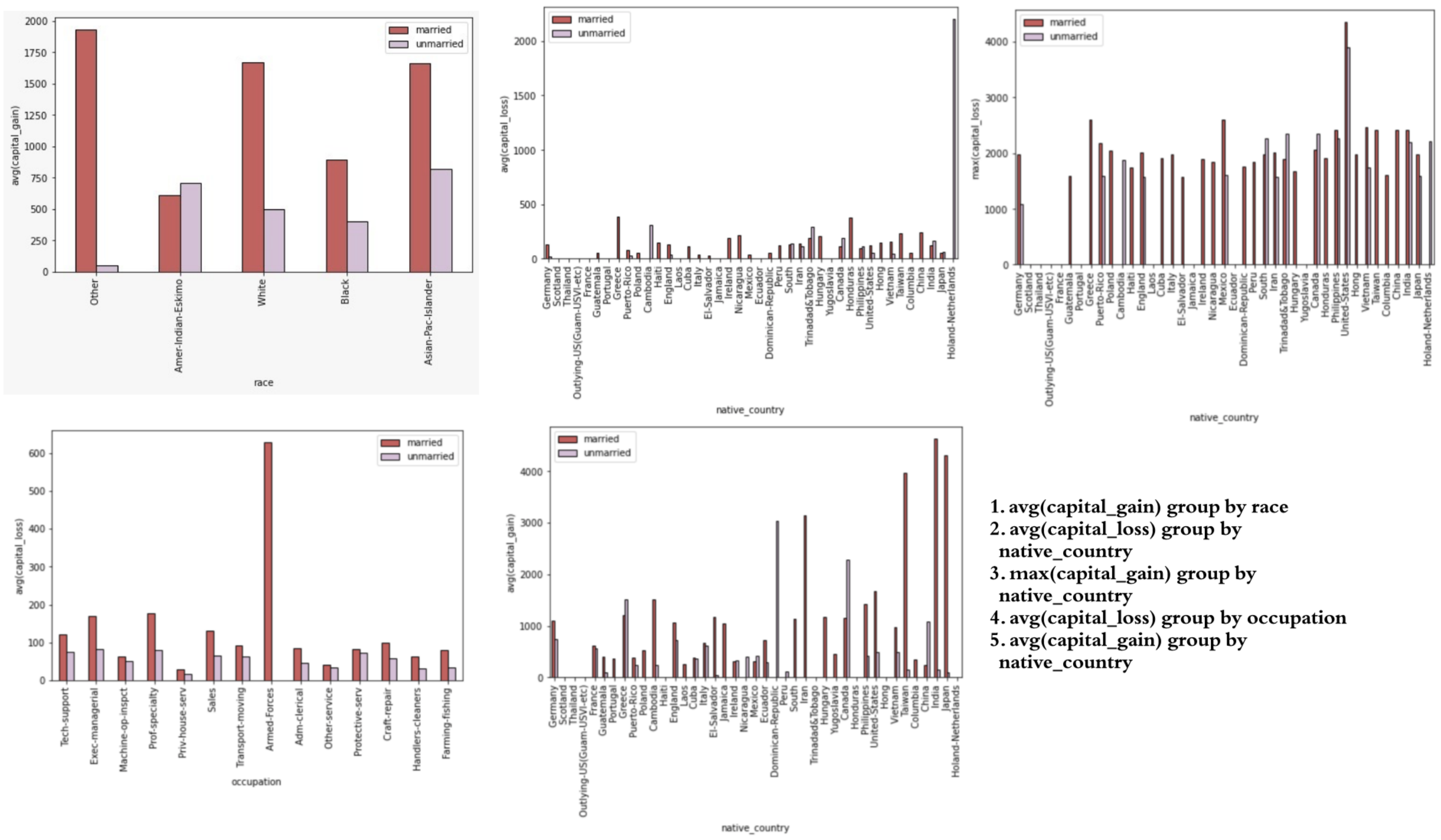
f. The output of the previous step, i.e., the final pruned set of views, is used to plot the top K views.

3. Experimental Results

The top-5 recommendations for the base dataset are:

1. avg(capital_gain) group by race
2. avg(capital_loss) group by native_country
3. max(capital_gain) group by native_country
4. avg(capital_loss) group by occupation
5. avg(capital_gain) group by native_country

Below are the corresponding top-5 visualizations generated after implementing the base implementation. The exact same plots were generated for the sharing+pruning-based implementation as well.



1. avg(capital_gain) group by race
2. avg(capital_loss) group by native_country
3. max(capital_gain) group by native_country
4. avg(capital_loss) group by occupation
5. avg(capital_gain) group by native_country

Fig 1. Top-5 visualizations for the base implementation and optimized implementation.

The following plots were recommended by the algorithm based on the utility metric which measures the deviation between the target and the reference view queries. Here, reference = 'married' and target = 'unmarried'. For the first plot between avg(capital_gain) grouped by race, we find that almost all races have a common trend where the avg(capital_gain) is greater for the married people, except American-Indian-Eskimos, whose avg(capital_gain) is somewhat similar for both the married and unmarried people.

For the plot between avg(captial_loss) grouped by native country, we find that the unmarried people of the Holland-Netherlands have the highest capital loss in comparison to the rest of the world.

For the plot between max(capital_loss) grouped by native country, the married people of the United States have the maximum capital loss, while the unmarried people of the rest of the world seem to have a higher maximum capital loss.

For the plot between avg(capital_loss) grouped by occupation, we find that the married people suffer the highest avg(capital_loss), but it is significantly higher for the people working in the armed forces.

For the first plot between avg(capital_gain) grouped by native country, we find that the married people of India have the highest avg(capital_gain), while almost all other countries have a higher avg(capital_gain) for the unmarried people.

3.2 Performance Analysis

We also compared our results by doing the performance evaluation to find out how well the sharing and pruning optimizations improve latency, and how the pruning optimizations affect accuracy. The primary evaluation metric used to compare the performance is *latency*. The total time for the base implementation without optimization was found to be 1.094 seconds, while the total combined time for the sharing and pruning-based optimization for the base implementation was found to be 0.282 seconds. A performance gain of almost 3.8x-4x is observed after optimization.

4. Extension and Results

4.1 Dataset

We have implemented this algorithm on a new dataset as part of our extension for this project. It is a survey dataset that can be used to assess the mental health of people working in the technological industry. The dataset has been obtained from Kaggle [3] and originally contains 27 features and 1259 data points. The following are the attributes of this dataset:

- Timestamp
- Age
- Gender
- Country
- state: If you live in the United States, which state or territory do you live in?
- self_employed: Are you self-employed?
- family_history: Do you have a family history of mental illness?
- treatment: Have you sought treatment for a mental health condition?
- work_interfere: If you have a mental health condition, do you feel that it interferes with your work?
- no_employees: How many employees does your company or organization have?
- remote_work: Do you work remotely (outside of an office) at least 50% of the time?
- tech_company: Is your employer primarily a tech company/organization?
- benefits: Does your employer provide mental health benefits?
- care_options: Do you know the options for mental health care your employer provides?
- wellness_program: Has your employer ever discussed mental health as part of an employee wellness program?
- seek_help: Does your employer provide resources to learn more about mental health issues and how to seek help?
- anonymity: Is your anonymity protected if you choose to take advantage of mental health or substance abuse treatment resources?
- leave: How easy is it for you to take medical leave for a mental health condition?
- mental*health*consequence: Do you think that discussing a mental health issue with your employer would have negative consequences?
- phys*health*consequence: Do you think that discussing a physical health issue with your employer would have negative consequences?
- coworkers: Would you be willing to discuss a mental health issue with your coworkers?
- supervisor: Would you be willing to discuss a mental health issue with your direct supervisor(s)?
- mental*health*interview: Would you bring up a mental health issue with a potential employer in an interview?
- phys*health*interview: Would you bring up a physical health issue with a potential employer in an interview?
- mental*vs*physical: Do you feel that your employer takes mental health as seriously as physical health?
- obs_consequence: Have you heard of or observed negative consequences for coworkers with mental health conditions in your workplace?
- comments: Any additional notes or comments

The answers to these questions give us a feel about the possible mental health condition of the people working in the tech industry. It can also be utilized to answer some thought-provoking questions like:

1. How does the frequency of mental health illness and attitudes towards mental health vary by geographic location?
2. What are the strongest predictors of mental health illness or certain attitudes towards mental health in the workplace?

4.2 Data Pre-Processing

The dataset was thoroughly cleaned before the algorithm could be applied to it to generate visualizations. The major steps involved in the data pre-processing for this dataset are as follows:

1. **Taking care of missing values:**
   a. We first found the missing values by percentage and by column names and removed the column ‘comments’ as it contained 86.7% missing values.
   b. We removed the column ‘Timestamps’ as it was not relevant to needs.
   c. We removed the column ‘State’ as it only had values corresponding to ‘United States’ as ‘Country’ and had 40% values missing.
   d. The NULL values in the columns ‘self_emplyed’ and ‘work_interefer’ were replaced by ‘No’ and ‘Never’ respectively.
2. **Making data consistent**
   a. We found the unique values in all the attributes and the bizarre values found for the attribute ‘Age’ were removed. They were [-29, 329, 99999999999, -1726, 5, 8, 11, -1]. Therefore, the rows with these values for ‘Age’ were dropped.
   b. The column ‘no_employees’ was renamed to ‘company_size’.
3. **Clubbing various attributes together into fewer attribute values**
   a. We clubbed the countries together and segregated them into six continents, namely, Asia, Africa, North America, South America, Oceania, and Europe.
   b. We clubbed all the gender values and segregated them into three genders, namely, male, female, and others.
   c. We clubbed the various values of the attribute ‘company_size’ and segregated them into four groups, namely, micro, small, medium, and large companies.

4.3 Methodology

The dataset only contained one numeric data type value, which is ‘Age’. We also implemented four sets of values for reference and target views, that is, for self_employed, family_history, treatment, and remote_work. The measure attributes M, and aggregate functions F were kept constant for all the four sets of target and reference view queries, and were [‘Age’] and [‘Average’, ‘Max’, ‘Min’, ‘Count’], respectively.

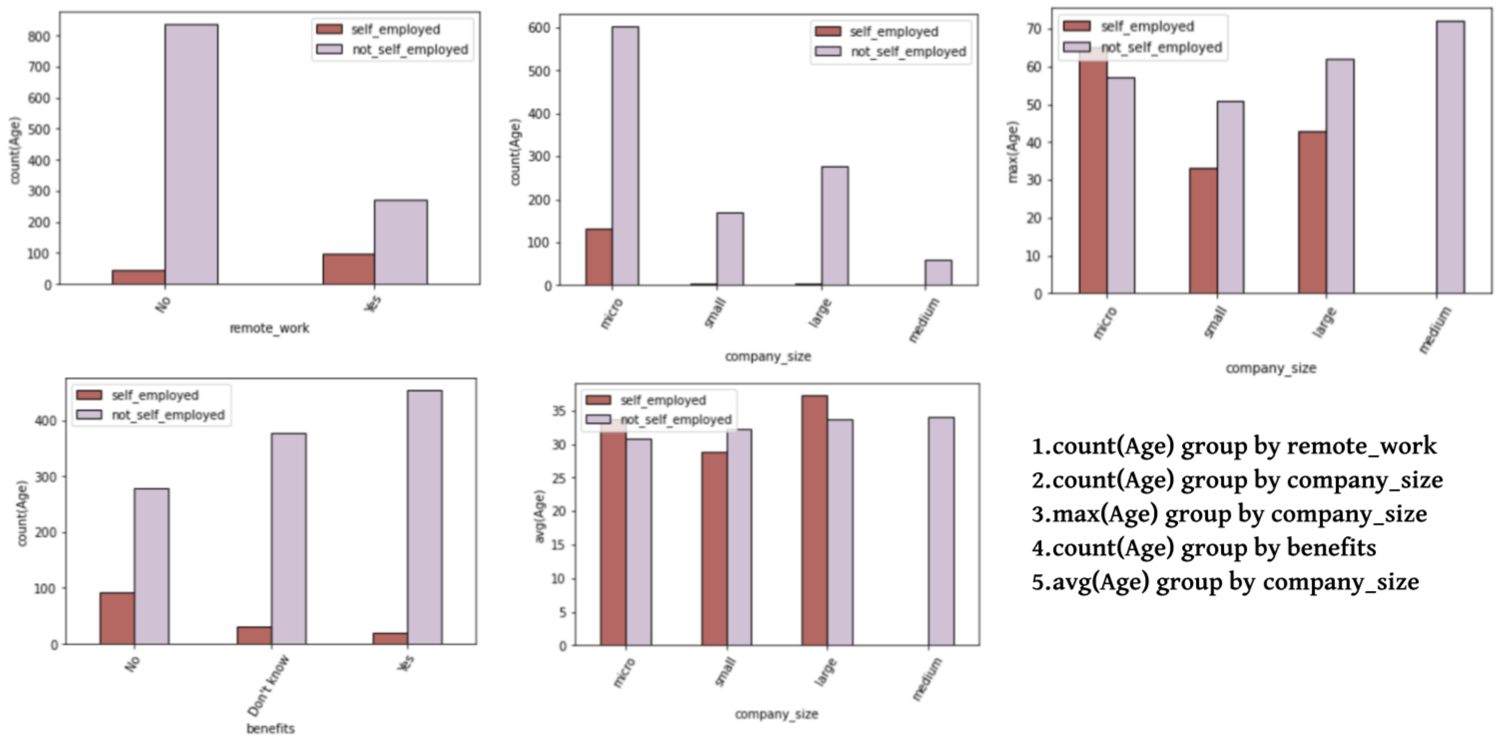
For the implementation of sharing and pruning-based optimization, the dataset was divided into ten chunks of equal size (more or less) each.

4.4 Results and Discussions

We successfully extended our implementation of the base model to the new dataset, namely, the Mental Health in Tech dataset. We implemented a total of 4 sets of target-reference view queries, that is, [self_employed, not_self_employed], [family_history, no_family_history], [treatment, no_treatment] and [remote_work, no_remote_work].

Therefore, a total of 4*2 = 8 tables were generated, and four sets of values were defined for Dimension Attributes A. The top-5 recommendations for the visualizations on this dataset are:

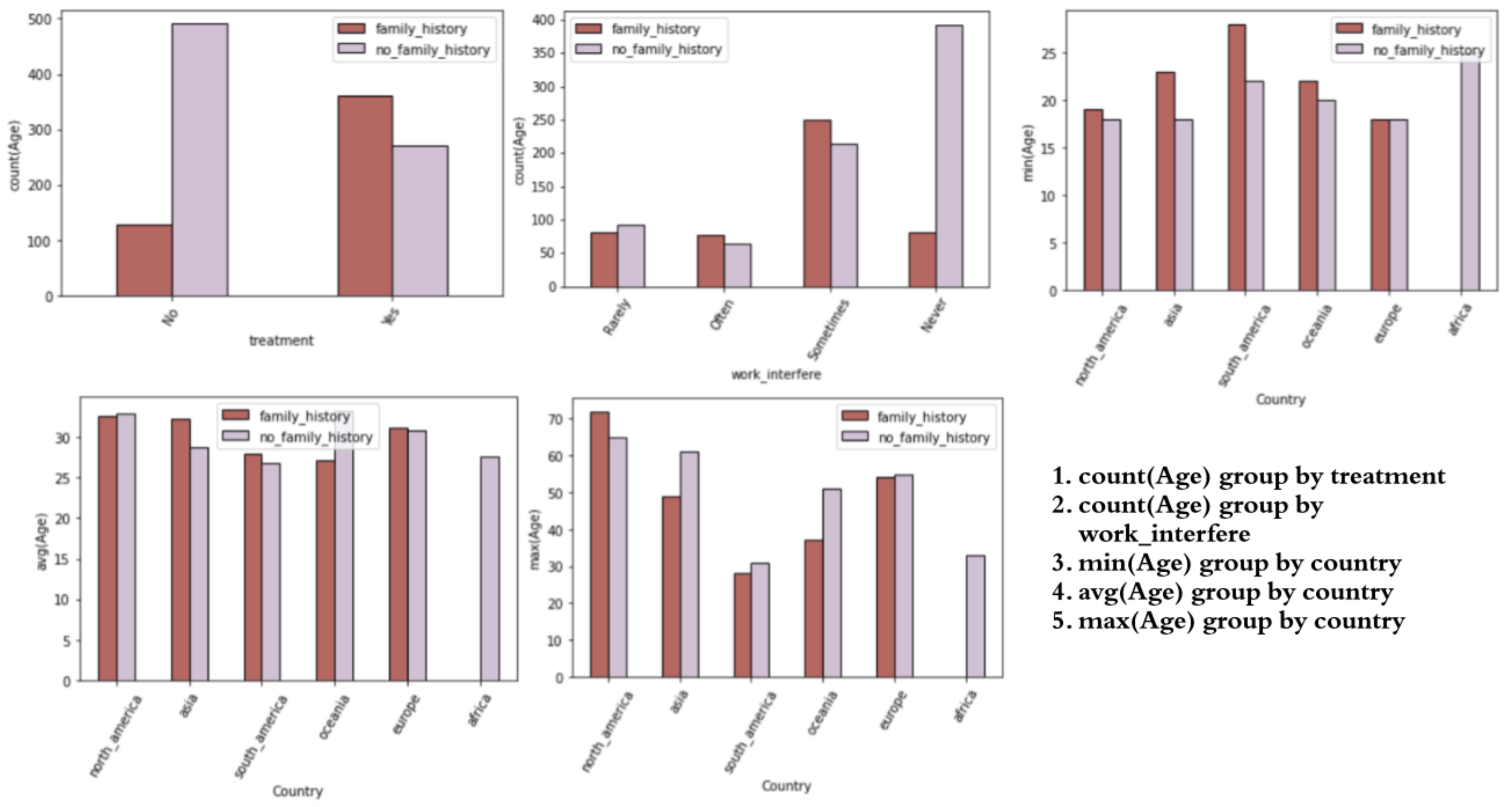
1. For reference-target views: ['self_employed_people', 'not_self_employed_people']



1.count(Age) group by remote_work
2.count(Age) group by company_size
3.max(Age) group by company_size
4.count(Age) group by benefits
5.avg(Age) group by company_size

Fig 2. Top-5 visualizations for the extension implementation for target-ref pair = self_employed, not_self_employed.

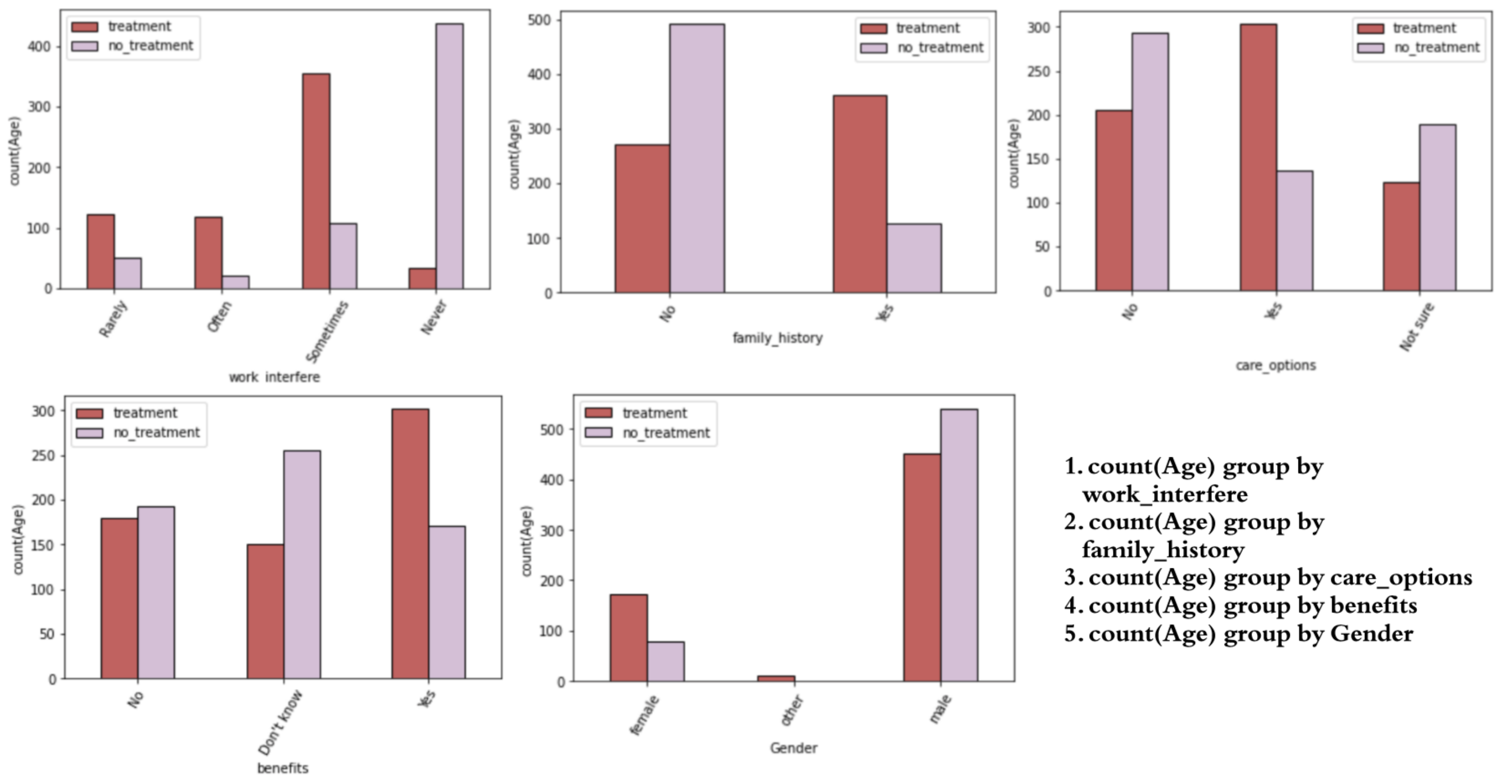
2. For reference-target views: ['family_history_people', 'no_family_history_people']



1. count(Age) group by treatment
2. count(Age) group by work_interfere
3. min(Age) group by country
4. avg(Age) group by country
5. max(Age) group by country

Fig 3. Top-5 visualizations for the extension implementation for target-ref pair = family_history, no_family_history.

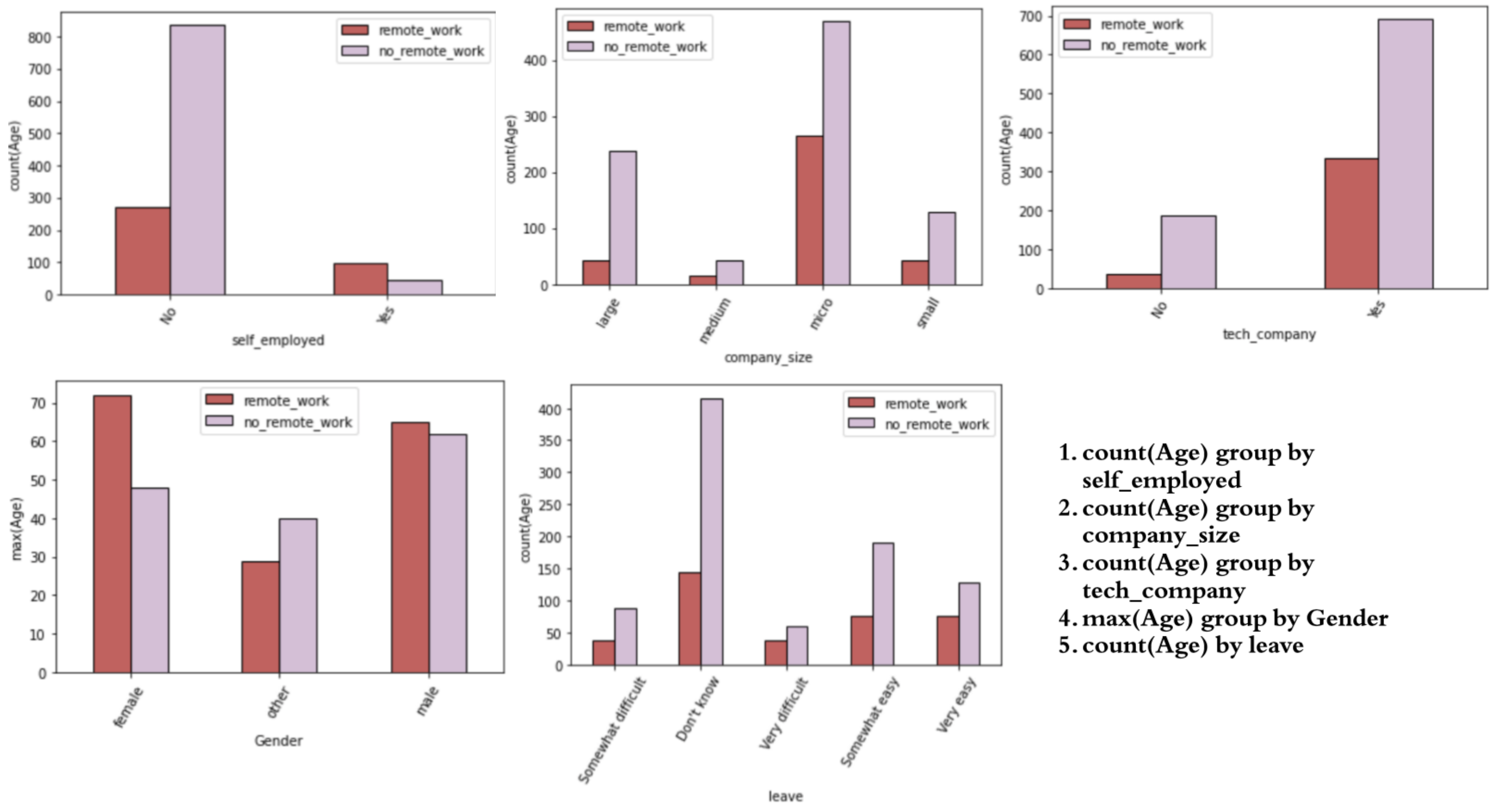
3. For reference-target views: ['treatment_people', 'no_treatment_people']



1. count(Age) group by work_interfere
2. count(Age) group by family_history
3. count(Age) group by care_options
4. count(Age) group by benefits
5. count(Age) group by Gender

Fig 4. Top-5 visualizations for the extension implementation for target-ref pair = treatment, no_treatment.

4. For reference-target views: ['remote_work_people', 'no_remote_work_people']



1. count(Age) group by self_employed
2. count(Age) group by company_size
3. count(Age) group by tech_company
4. max(Age) group by Gender
5. count(Age) by leave

Fig 5. Top-5 visualizations for the extension implementation for target-ref pair = remote_work, no_remote_work.

It is clear from the plots that the algorithm picks out the most interesting plots based on the deviation. For reference-target views: ['self_employed_people', 'not_self_employed_people'], the most interesting plot has been recommended as count(Age) group by remote work as it has the highest amount of deviation between the target and the reference views. The key insight from the most interesting plot is that the maximum number of people are not working remotely, and they are not self-employed. That is, companies are making their employees work from the office.

For reference-target views: ['family_history_people', 'no_family_history_people'], the most interesting plot has been recommended as count(Age) by treatment. It displays the count of people who have sought treatment for mental health issues and if they have a family history of mental problems or not. The plot shows that the maximum number of people who don't seek treatment, do not have a family history, and the maximum number of people who do seek treatment have a family history of mental issues.

For reference-target views: ['treatment_people', 'no_treatment_people'], the most interesting plot has been recommended as count(Age) by work interfere. It displays the count of people who feel that work (often, rarely, sometimes, and never) interferes with their mental health, and how many of them seek treatment. The plot shows that the maximum number of people who feel that their work is not interfering with their mental health are not seeking treatment for it, while the maximum number of people who feel that their work sometimes interferes with their work are seeking treatment.

For reference-target views: ['remote_work_people', 'no_remote_work_people'], the most interesting plot has been recommended as count(Age) by company size. It displays the count of people who work at companies of different sizes and who all are working remotely or not. The plot shows the maximum number of people who work at micro-companies are not working remotely.

We performed sharing-based and pruning optimization for the extension dataset as well, and measured the performance using latency as the performance metric. For the non-optimized extension implementation, we found that the ref-target pairs ['self_employed_people', 'not_self_employed_people'], ['family_history_people', 'no_family_history_people'], ['treatment_people', 'no_treatment_people'] and ['remote_work_people', 'no_remote_work_people'] took 0.3296 seconds, 0.079 seconds, 0.061 seconds and 0.068 seconds respectively. It was found that total combined execution time for sharing and pruning-based optimization for the extension dataset with ref-target pair ['self_employed_people', 'not_self_employed_people'] was found to be 0.244 seconds. A performance gain of upto 1.6x was observed.

5. Summary

In this project, we have successfully replicated the original results obtained by the authors of SeeDB and have also performed sharing and pruning-based optimization techniques to enhance the efficiency and latency of the system. We were not only able to improve latency but also reduced the waste of computational resources by performing sharing-based optimization where we combined multiple aggregate view queries into a single view query as all of them were being applied for the same underlying dataset.

We even extended our base implementation to a new dataset, the Mental Health in Tech dataset, and also performed sharing and pruning-based optimizations for this dataset as well. For the base implementation of the extension dataset, we tested for 4 sets of reference-target views, namely, [self_employed, not_self_employed], [family_history, no_family_history], [treatment, no_treatment] and [remote_work, no_remote_work], and plotted the top-5 visualizations for all of them. The best, which is the target-reference pair having the highest deviation value was picked and both the optimizations were performed on this target-reference pair.

The performance of this implementation was evaluated using latency as the metric, and we measured the combined time for both optimizations (sharing+pruning) together. The total combined time for the base implementation was found to be 0.282 seconds, while the total combined time for the extension implementation was found to be 0.244 seconds.

References:

1. Vartak, M., Parameswaran, A., Polyzotis, N., & Madden, S. R. (2014). SEEDB: automatically generating query visualizations.
2. https://archive.ics.uci.edu/ml/datasets/Census+Income.
3. https://www.kaggle.com/datasets/osmi/mental-health-in-tech-survey