# Reproducing partial results from SeeDB: efficient data-driven visualization recommendations to support visual analytics

**645 Mini-Project Report**

*Amuktha Badimala, Sahithi Singireddy, Sai Mahesh Sure, Sravani Gona*

**Github link to code repository**: https://github.com/SravaniGona/SeeDB-implementation

## 1. Introduction:

This project focuses on reproducing partial results from the research paper "SeeDB: efficient data-driven visualization recommendations to support visual analytics" [1]. In this paper the authors propose a SeeDB, a visualization recommendation engine to facilitate fast visual analysis.

They proposed Sharing-based Optimizations to intelligently merge and batch queries, reducing the number of queries issued to the database and, in turn, minimizing the number of scans of the underlying data and Pruning-Based Optimizations to quickly identify high-utility visualizations we adopt a deviation based metric for visualization utility, while indicating how we may be able to generalize it to other factors influencing utility. We implemented the algorithm based on these Optimizations to find top-5 aggregate views on the Census dataset [2].

## 2. Problem Statement:

Given a user-specified query to include married people on Census database, a reference query to include unmarried people and Kullback-Leibler Divergence (K-L divergence) as the utility function, find the top 5 aggregate views by the utility measure among all the views while implementing the Shared-based Optimization through query rewriting and Pruning-based Optimization using Hoeffding-Serfling inequality.

## 3. Dataset:

The Census Income dataset was extracted by Barry Becker from the 1994 Census database and was originally compiled for predicting whether a person makes 50k a year. The dataset consists of 32561 training instances and 16281 testing instances. It contains both numerical attributes such as age, capital gain, capital loss, etc and categorical attributes such as workclass, race, relationship, etc. We considered both training and testing instances for implementing this paper.

## 4. Methodology:

### Preprocessing:

The data is provided in two files, one for training instances and one for testing instances. We created a Postgres database “census” with a table named “adults” to store this data, with columns corresponding to the features in the dataset. Both files were imported into the “adults” table. Upon loading the data, the columns with missing values were set to NULL.

### Assumptions:

As a requirement of our project to compare visualizations between married and unmarried individuals, we needed to transform the data. The original data's "marital-status" column included various values such as divorced, separated, and widowed. To ensure a balanced and fair comparison, we decided to categorize the data into two groups: married and unmarried. The categorization is as follows:

Married: Married-civ-spouse, Married-spouse-absent, Married-AF-spouse, Separated
Unmarried: Divorced, Never-married, Widowed

### Implementation:

- **Finding all aggregate views:** Following the approach outlined in the paper, we define three lists, a, m, and f, consisting of attributes and possible aggregate functions as below and we compute all combinations of (a,f,m) lists.

```
# Aggregate functions
f_list = ["count","sum", "min", "max", "avg"]

# Dimension attributes

a_list  =  ["workclass",  "education",  "occupation",  "race",
"sex", "relationship", "native_country", "salary_range"]

# Measure attributes
m_list  =  ["age",  "fnlwgt",  "education_num",  "capital_gain",
"capital_loss", "hours_per_week"]
```

- **Shared-based Optimization:**

  The exhaustive approach requires the execution of f×a×m queries on the database for user-specified query and again for reference query. This often involves grouping by the same attribute multiple times to calculate different aggregate values on different columns. Hence we use sharing-based optimization to address this issue.

  By **combining multiple aggregates**, we have rewritten the query with multiple aggregations and the same group-by attribute to retrieve a set of aggregates on multiple column values instead of one at a time. This significantly reduces the number of queries executed on the database.

  We have also **combined user-specified and reference view query**, and split the results based on the marital_status.

  **Additional Implementation:**

  In addition to these, we **combined multiple group-by clauses**. We generated a single combined query consisting of all possible aggregates and grouping by all attributes in 'a_list'. However, grouping by more than one attribute posed challenges in retrieving results for a single value of a column. For instance, when grouping by A & B, column results appeared in the form of (1, 2) and (1, 3), where the first aggregated value corresponds to A and the second corresponds to B, resulting in multiple tuples with A as 1.

  To address this issue, we implemented an aggregation mechanism to retrieve corresponding tuples for a column value and then used multiple aggregate values to calculate a combined aggregation value like sum of sums, max of max, min of min and sum of count. Calculating the aggregated average required using the counts and sums of that column to divide the total sum by total count and obtain the overall average per category of 'a'.

- **Pruning-based Optimization:** We adopted confidence interval-based pruning to optimize our visualization recommendation process. Initially, we divided the database into 10 subsets. On each subset, we executed the shared-based optimization query, computing the worst-case confidence intervals using the Hoeffding-Serfling inequality. Views with low utility were pruned according to the following rule: if the upper bound of the utility of view Vi was lower than the lower bound of the utility of five or more views, Vi was discarded. During the first iteration, we refrained from pruning low utility views due to a mathematical error caused by the log(log(m)) function. After each iteration, we recomputed utility scores, mean utility, and confidence intervals. This iterative process led to a gradual decline in the number of views until only a subset remained. At the end of our 10 iterations, we selected the top 5 views based on the KL Divergence utility measure.

- **Handling aggregate results:** When splitting the aggregate results based on marital status, we encountered an issue where the results had different group-by indexes for each attribute. To compute KL divergence between these two sets of results, we needed them to be symmetric. Simply inserting the appropriate missing indexes with a 0 value was not possible, as we primarily used KL divergence as our distance measure for calculating utility. Including a value of 0 in KL divergence calculations would result in infinity.

  To address this issue and maintain result integrity, we implemented alpha smoothing. Each column value with a missing or zero aggregate value was replaced with a very small value (1e-10). After traversing the results of each query, we normalized them to scale each result between 1 and 0. Subsequently, we calculated the utility for each view.

## 5. Results:

The implemented algorithm successfully generated interesting visualizations for the census database. The top-5 interesting visualizations are observed when the data is grouped-by relationship and aggregated as sum of fnlwgt, capital gain, capital loss, hours per week and age. They achieved the highest K-L Divergence scores/entropy scores (SciPy), indicating significant differences that would be reflected in the corresponding visualizations.

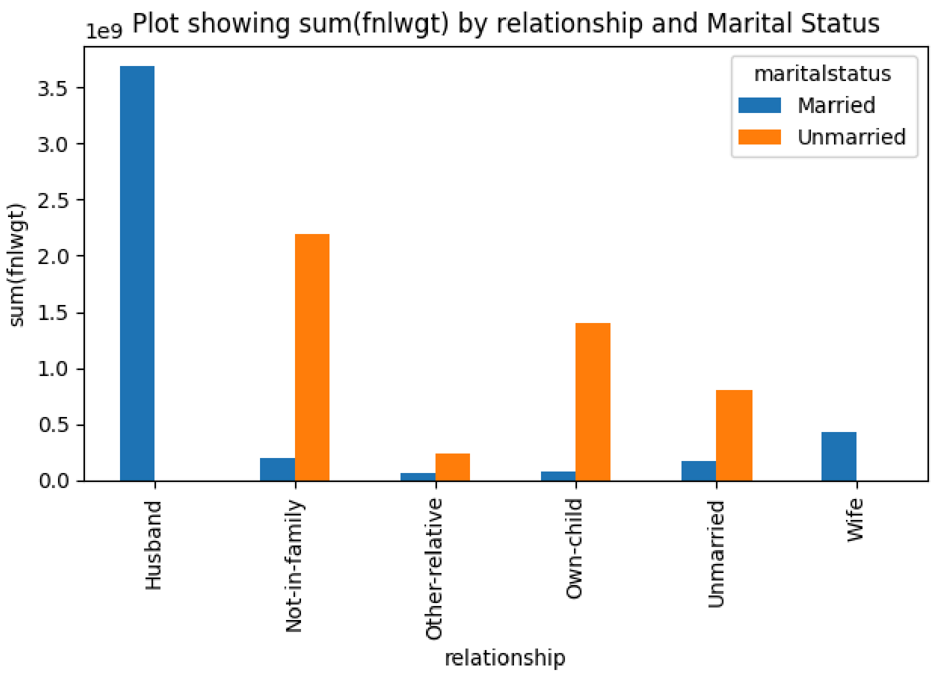

*Figure 1: Interesting visualization of sum of fnlwgt vs relationship*

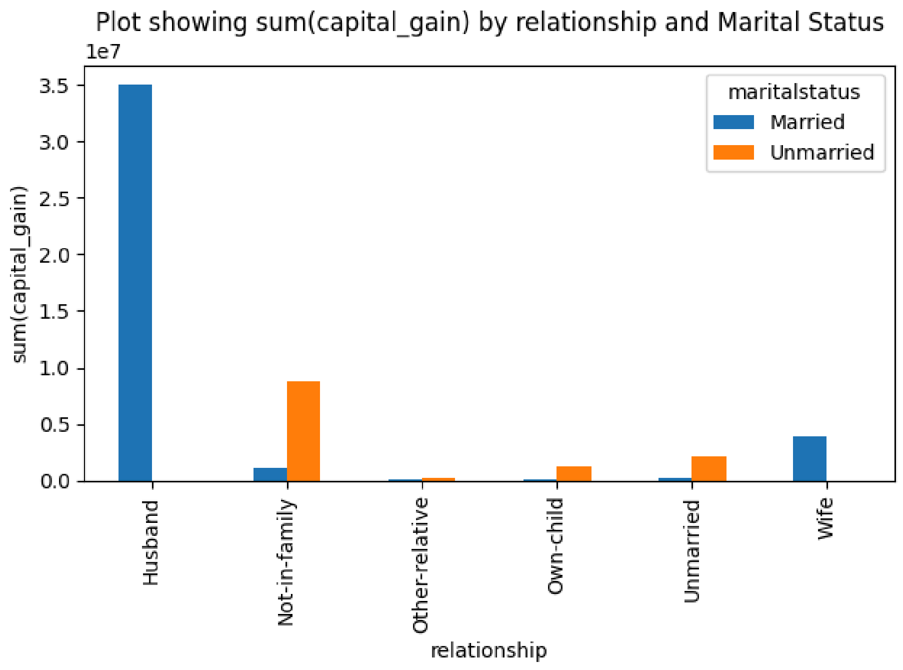

*Figure 2: Interesting visualization of sum of Capital gain vs relationship*

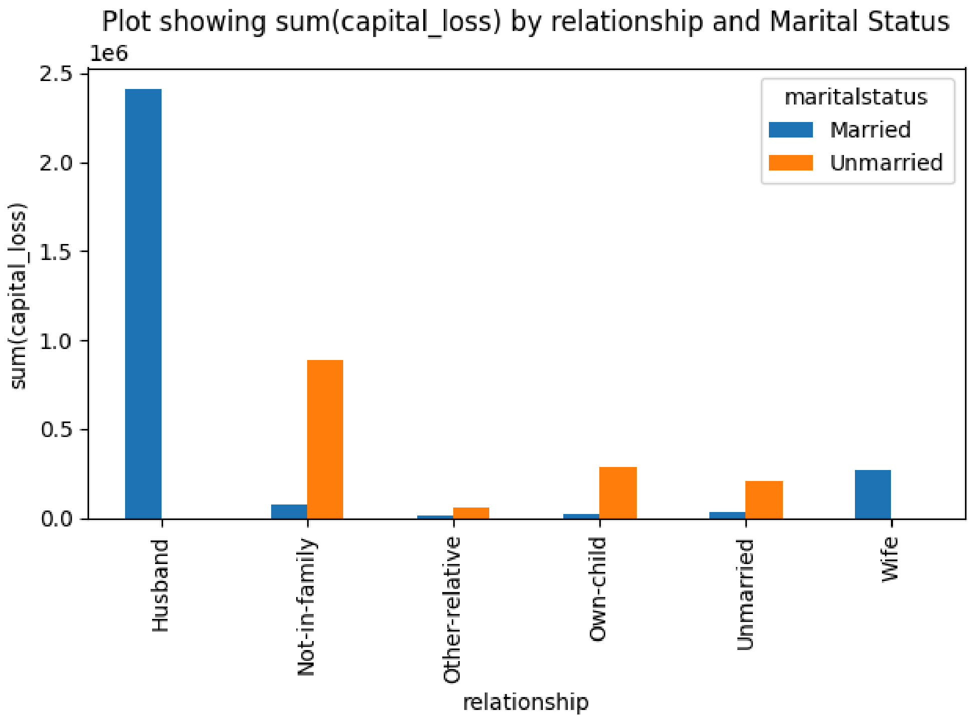

*Figure 3: Interesting visualization of sum of Capital Loss vs relationship*

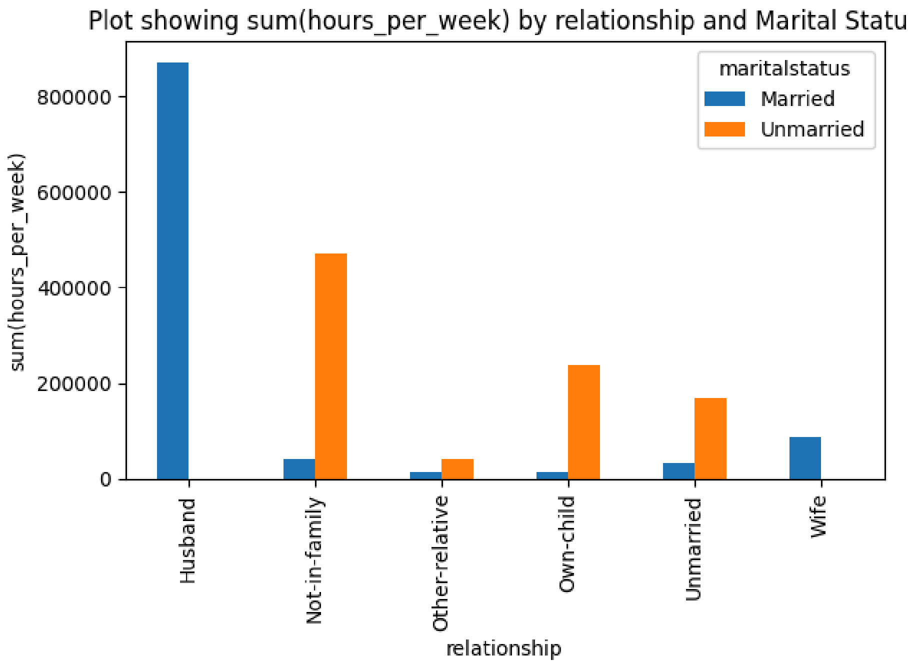

*Figure 4: Interesting visualization of sum of Hours per week vs relationship*

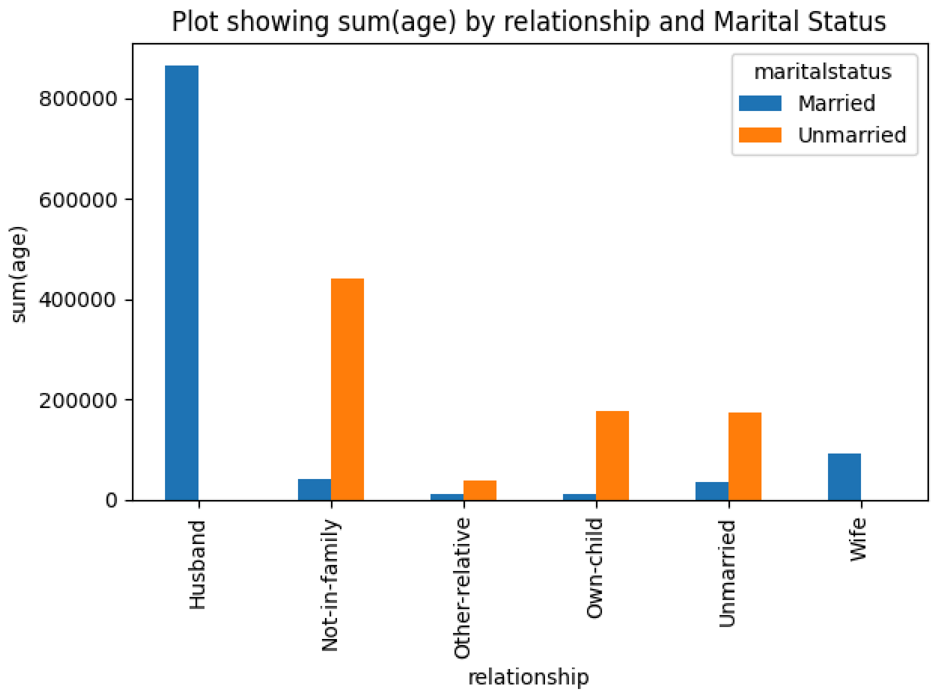

*Figure 5: Interesting visualization of sum of Age vs relationship*

**Interpretation:**

For married individuals, the highest sum of features is observed for those identified as husbands, followed by wives, not-in-family, other relatives, own children, and unmarried individuals. For unmarried individuals, the highest sum of features is observed for those identified as not-in-family, followed by unmarried individuals, own children, other relatives, and a negligible sum for husbands and wives.

These visualizations help to understand how the sum of features varies based on both marital status and relationship. For instance, it appears that the highest sum of features is typically associated with individuals identified as husbands, followed by other categories, while the sum of features for unmarried individuals is lower across all relationship categories.

## 6. Conclusion:

This project successfully reproduced core functionalities of SeeDB, a system for recommending data visualizations. By implementing the algorithm using Shared-based Optimization through query rewriting and Pruning-based Optimization using Hoeffding-Serfling inequality, we generated interesting visualizations based on user queries and data characteristics. The project demonstrates the potential of such systems in aiding data exploration and knowledge discovery.

## 7. Additional Implementation:

In addition to these, we **combined multiple group-by clauses**. We generated a single combined query consisting of all possible aggregates and grouping by all attributes in 'a_list'. However, grouping by more than one attribute posed challenges in retrieving results for a single value of a column. For instance, when grouping by A & B, column results appeared in the form of 1, 2 and 1, 3, where the first aggregated value corresponds to A and the second corresponds to B, resulting in multiple tuples with A as 1.

To address this issue, we implemented an aggregation mechanism to retrieve corresponding tuples for a column value and then used multiple aggregate values to calculate a combined aggregation value like sum of sums, max of max, min of min and sum of count . Calculating the aggregated average required using the counts and sums of that column to divide the total sum by total count and obtain the overall average per category of 'a'.

## 8. Contributions:

All of us contributed to the overall design and planning of the project, including defining the problem statement and methodology.

Amuktha and Mahesh:

- Conducted literature review and provided insights into reproducing partial results from the research paper.
- Conducted preprocessing of the dataset, including importing training and testing instances into the Postgres database and handling missing values.
- Defined lists of attributes and possible aggregate functions (a_list, m_list, f_list) and computed all combinations of these lists to find all aggregate views.
- Contributed to the selection of top 5 visualizations based on KL Divergence scores and assisted in the interpretation of results.
- Assisted in code review, debugging, and documentation of the project.

Sahithi and Sravani:

- Implemented the shared-based optimization through combining multiple aggregates, Group-BYs, target and reference query to reduce the number of queries issued to the database.
- Developed the code for handling aggregate results, including the implementation of alpha smoothing to address missing or zero aggregate values.
- Implemented the pruning-based optimization, dividing the database into subsets, computing confidence intervals using Hoeffding-Serfling inequality, and pruning views with low utility.
- Assisted in code review, debugging, and documentation of the project.

## 9. References:

**[1]** M. Vartak, S. Rahman, S. Madden, A. Parameswaran and N. Polyzotis, "SeeDB: Efficient data-driven visualization recommendations to support visual analytics", Proceedings of the VLDB Endowment, vol. 8, no. 13, pp. 2182-2193, 2015.

**[2]** "UCI Machine Learning Repository: Census Income Data Set", Archive.ics.uci.edu, 2018. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Census+Income.