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

Lingxi Li
University of Massachusetts Amherst
lingxili@umass.edu

Yuefeng Zhang
University of Massachusetts Amherst
yuefengzhang@umass.edu

Junzhu Li
University of Massachusetts Amherst
junzhuli@umass.edu

Yiming Yuan
University of Massachusetts Amherst
yimingyuan@umass.edu

All authors have equal contributions

1 Introduction

In this replication report, we reproduced the implementation of SeeDB (Vartak et al., 2015), an efficient data-driven visualization recommendation system, from engineering perspective and tested its performance on Census Income dataset (Kohavi, 1996). The implementation, environment setup instruction, pre-processed data, jupyter notebooks, and evaluation script can be found in our public GitHub repository[1].

The first step in data analysis is always visualization. The information provided by the visualization is then analyzed in greater detail. However, we are faced with a vast amount of data, and the need to visualize it in an interesting way is not an easy task. On the one hand, we have to think about how to generate visualizations for the user within the interaction time, and on the other hand, we have to ask ourselves whether the visualizations generated so far are interesting or not.

To solve these two problems, SeeDB uses pruning and sharing optimizations to improve computational efficiency and achieve a tolerable interaction time for the user. In order to obtain relevant visualizations, SeeDB uses a deviation-based metric to measure which visualizations are of interest to the user. The final conclusion of the experiment is that with the above solution, SeeDB can recommend valuable data visualizations in the state-of-the-art running time.

Our replication work consists of three parts: 1) declaration of uncertain factors, 2) technical implementations, and 3) performance evaluation. The clear-out analysis of uncertain factors from the original problem will be defined in Section 2 and Section 3 for building a self-consistent mind model of SeeDB and explaining the reasoning behind our decisions. The architecture for our technical implementations will be described in Section 3, including pseudo code and detailed computation approaches. In the end, the observed performance and result alignment comparison will be discussed in Section 4.

2 Problem Analysis

Each SeeDB visualization can be represented as an aggregate over a group-by query on the underlying data. The structure of these view queries can be expressed as a triple, denoted as V_i, where each V_i consists of the following components:

$$V_i = (\underbrace{d}_{\text{Dimension}}, \underbrace{m}_{\text{Measure}}, \underbrace{f}_{\text{Aggregate Function}})$$

Here is an example of an SQL query used in our data processing:

```
SELECT d, f(m) FROM TABLE GROUP BY d WHERE selection-predicate
```

such that inline variables `d`, `m`, and `f` are iterated from Table 1.

Our problem is: in order to get the k most interesting views, for each aggregate view we need to generate two series based on the target and referee vies, so each requires $D \times M \times F$ potential visualizations, and two is $2 \times D \times M \times F$ queries to the DBMS. Each query may scan the whole table. And computation wasted on low-utility views.

[1]Public GitHub Repository: https://github.com/lilingxi01/seedb-replication

Components	Values
d	workclass, education, occupation, relationship race, sex, native-country, income
m	age, fnlwgt, education-num, capital-gain capital-loss, hours-per-week
f	count, max, min, sum, avg

Table 1: Candidate values for inline variables `d`, `m`, and `f` in our core SQL statement discussed in Section 2. In non-optimizing setup, all queries are made by iterating through combinations of candidate values from three components onto core SQL statement.

3 Implementations

3.1 Data Pre-processing

Our task was to compare the Socioeconomics of married and unmarried adults using Census data. The attributes analyzed include: *age*, *education*, *marital-status*, *sex*, *race*, *income*, *hours-worked*, etc. Initially, we import `adult.data`, which contains 32,561 rows and 15 columns. This data is processed using the pandas library to handle question marks and spaces. The *marital-status* column is uniquely extracted and printed for analysis. A mapping dictionary, `matrital_status_mapping`, is created, and the marital states are divided into `married` and `unmarried` states according to the extracted marital states. The marital status in the original data is altered to `married` and `unmarried`, facilitating future analysis and extraction. Finally, the processed data is exported.

The attributes database table is based on the raw data, and the data just processed for export is imported into the table. The data in the `adult_data` table is then distributed into different database tables based on `married` and `unmarried` status as conditions.

3.2 Data Partitioning

Data partitioning is used for supporting the pruning-based optimizations mentioned in Section 3.5, where each data partition is applied through all remaining candidate views in each phase defined in the original paper. We randomly partition the original Census Income dataset (Kohavi, 1996) into 5 partitions, each containing 6514 rows.

Due to the static nature of this dataset, we partition data during pre-processing by assigning each row with a partition key (from 0 to 4). All rows with the same partition key belong to the same partition, and this column is filtered at runtime during query combinations discussed in Section 3.4. In order to improve the runtime performance, we created an index on the column of partition key for query efficiency.

3.3 Kullback-Leibler Divergence

Kullback-Leibler Divergence, or K-L divergence, is a method of quantifying the difference between two probability distributions P and Q, also known as relative entropy. In the fields of probability and statistics, it is common to use a simpler, approximate distribution instead of observational data or a distribution that is too complex to handle directly. K-L divergence is a useful metric for measuring the amount of information lost when using one distribution to approximate another.

Suppose that for a random variable ξ, there are two probability distributions P and Q. If ξ is a discrete random variable, the KL divergence from P to Q is defined as:

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

where $\sum_x$ denotes the sum over all possible outcomes of ξ, $P(x)$ is the probability of x according to P, and $Q(x)$ is the probability of x according to Q. It is important to note that $\log$ denotes the natural logarithm, and this calculation assumes that $P(x) > 0$ and $Q(x) > 0$ for all x in the domain of P.

3.4 Sharing-based Optimizations

SEEDB evaluates very similar queries that scan the same underlying data and differ only in grouping and aggregation attributes, aiming to reduce the number of queries issued to the database and minimize data scans. It involves optimizations such as query merging, batch processing, common sub-expression elimination, and intermediate result reuse, which enhance efficiency by reducing redundant computations and improving overall system performance. We implemented the following optimizations:

Combine Multiple Aggregates: The goal is to minimize database scans by combining multiple aggregate queries that share the same group-by attribute into a single query with multiple aggregations. For example, instead of running separate queries like:

```
Q1 = SELECT a, f(m1) FROM D GROUP BY a
Q2 = SELECT a, f(m2) FROM D GROUP BY a
```

Instead of executing two separate queries for each view, we can combine these views into a single query:

```
Q3 = SELECT a, f(m1), f(m2) FROM D GROUP BY a
```

This implementation combines the queries into one view, reducing the overall number of queries that need to be executed. In this example, the combined query only requires one execution instead of two.

To measure the effectiveness of the implemented optimization, we compared the execution times of the baseline approach (executing individual queries) and the optimized approach (executing combined queries). The results (listed in Section 4.2) demonstrate a significant improvement when using the sharing-based optimization.

3.5 Pruning-based Optimizations

Confidence-interval pruning algorithm relies on the Hoeffding-Serfling inequality (Serfling, 1974) for providing a confidence interval around the estimated mean of utility score over any view V_i as described in the original technical report (Vartak et al.).

$$\Pr\left[\exists m, 1 \le m \le N : \left|\frac{\sum_{i=1}^{m} Y_i}{m} - \mu\right| > \varepsilon_m\right] \le \delta$$

This equation consists the following parts that were not clearly defined in the original paper (Vartak et al., 2015) and we re-define them here for clarification and experiment consistency:

- The hyperparameter δ controls the error tolerance when computing the confidence interval, thus the smaller the better. For consistency, we define $\delta = 0.0001$ in this replication work as this value is low enough for our measurement. When δ gets even lower, the number of views kept after each phase will be decreased.

- The variable m and N in confidence-interval pruning method reflect the sampling size and the actual size when computing the mean utility over any view V_i. Since we partition the data into 5 partitions (as discussed in 3.2), we always have $N = 5$. And because views could potentially be dropped after pruning in the end of each phase, we will not get utility scores on all data partitions for all views (as discussed in Section 3.3), thus m represents the sample size when estimating the actual utility, which is equivalent to the number of data partitions that a view V_i has been run over. In other words, we collected a partial utility score from each data partition u on a view V_i as a sample, and the number of collected partial utility scores on V_i is the sample size m for V_i, as shown in the following equation where C_u is a list of candidate views for phase u and $\mathbb{1}$ is a binary tester.

$$m = \sum_{i=1}^{N} \mathbb{1}\{V_i \in C_u\}$$

After that, we can then compute the confidence interval $\mathbb{CI} = [\mu' - \varepsilon_m, \mu' + \varepsilon_m]$ where $\mu' = \frac{\sum_{i=1}^{m} D_{KL}}{m}$ and ε_m can be computed with δ, m and N:

$$\varepsilon_m = \sqrt{\frac{(1 - \frac{m-1}{N})(2 \log \log(m) + \log(\pi^2/3\delta))}{2m}}$$

Due to the motivation nature of Y_i from SeeDB (Vartak et al., 2015), this equation can fail when $m = 1$ because $\log \log(1) = -\infty$. To resolve this error, we updated the equation as follow for achieving the optimal result (which perfectly aligned with our baseline results as discussed in Section 4.2) without failing the computation:

$$\bar{\varepsilon}_m = \sqrt{\frac{(1 - \frac{m-1}{N})(2 \log(m) + \log(\pi^2/3\delta))}{2m}}$$

In the end of each phase u, we drop candidate view V_i from C_u when the upper bound of V_i is lower than the k-th largest lower bound, and use all remaining candidate views in the next phase $u + 1$.

4 Deliverables and Evaluation Results

4.1 Top-5 Aggregate Views

The top-5 most interesting views from Census Income dataset (Kohavi, 1996) is listed in Table 2. Bar chart graph for top-5 most interesting views are attached below as Figure 1, 2, 3, 4, and 5.

Attribute	**Measurement**	**Aggregation**	**Score**
Relationship	Capital Gain	Sum	21.23
Relationship	Capital Loss	Sum	20.44
Relationship	Hours per Week	Sum	19.31
Relationship	Education Number	Sum	19.16
Relationship	Age	Count	18.81

Table 2: KL-Divergence Score for top-5 utility views.

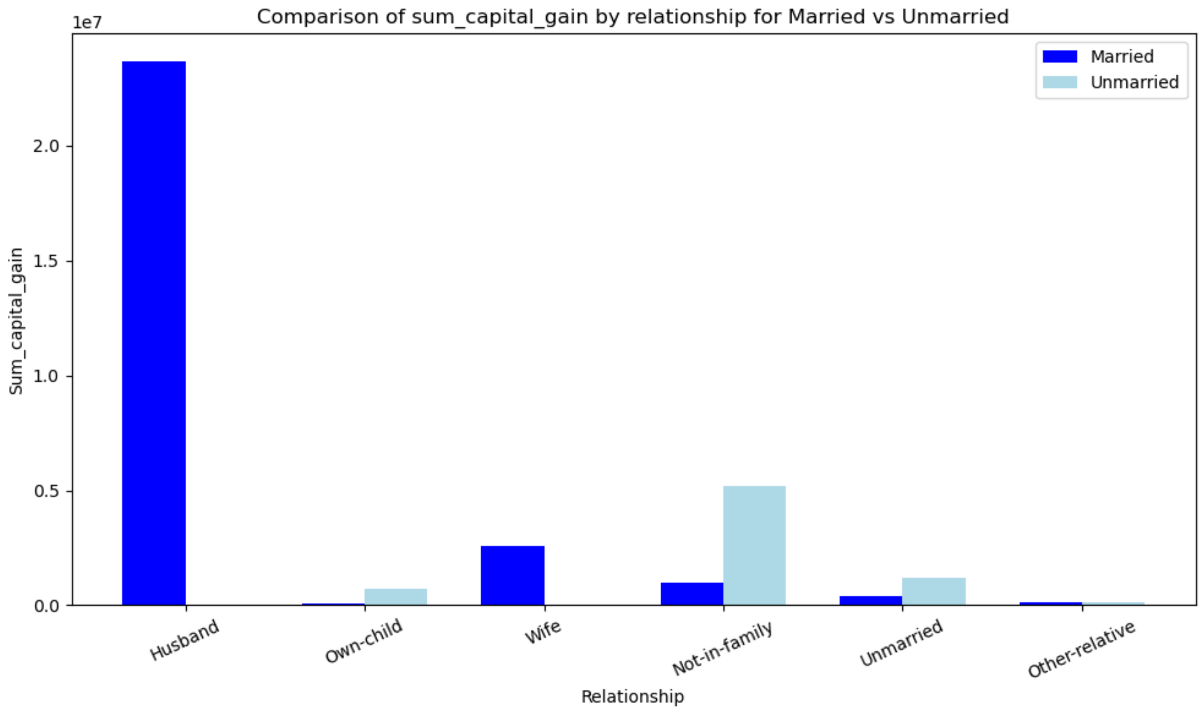

Figure 1: Sum of capital gain by relationship.

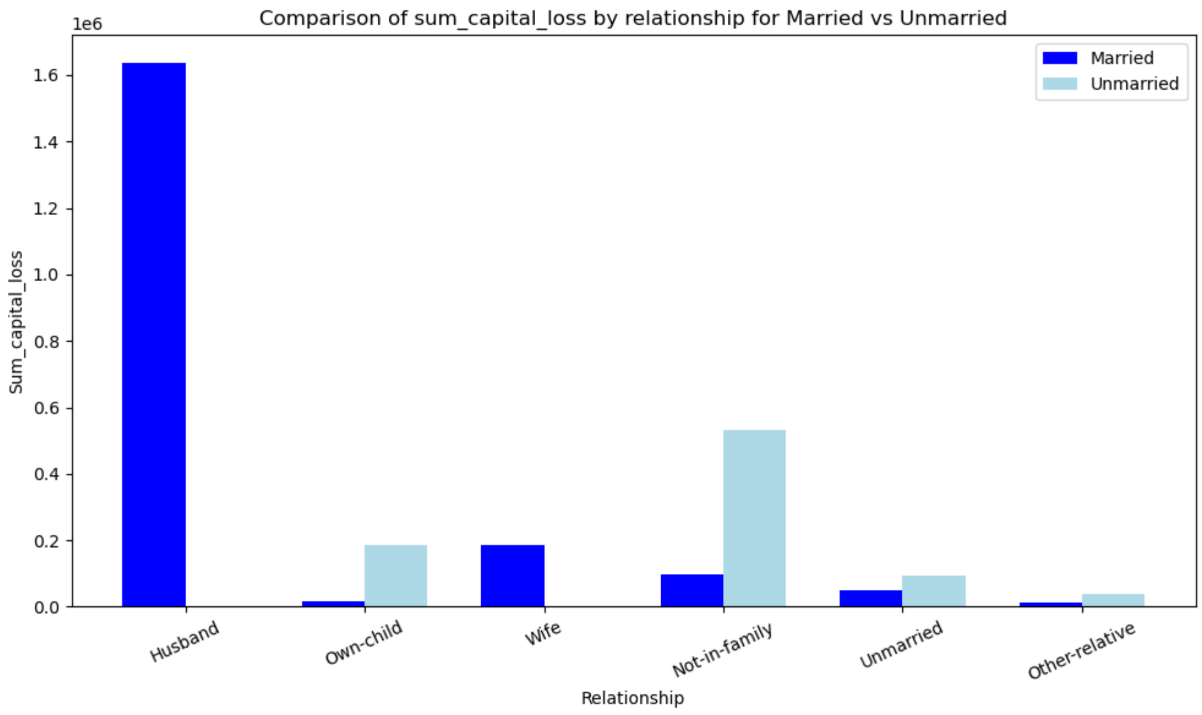

Figure 2: Sum of capital loss by relationship.

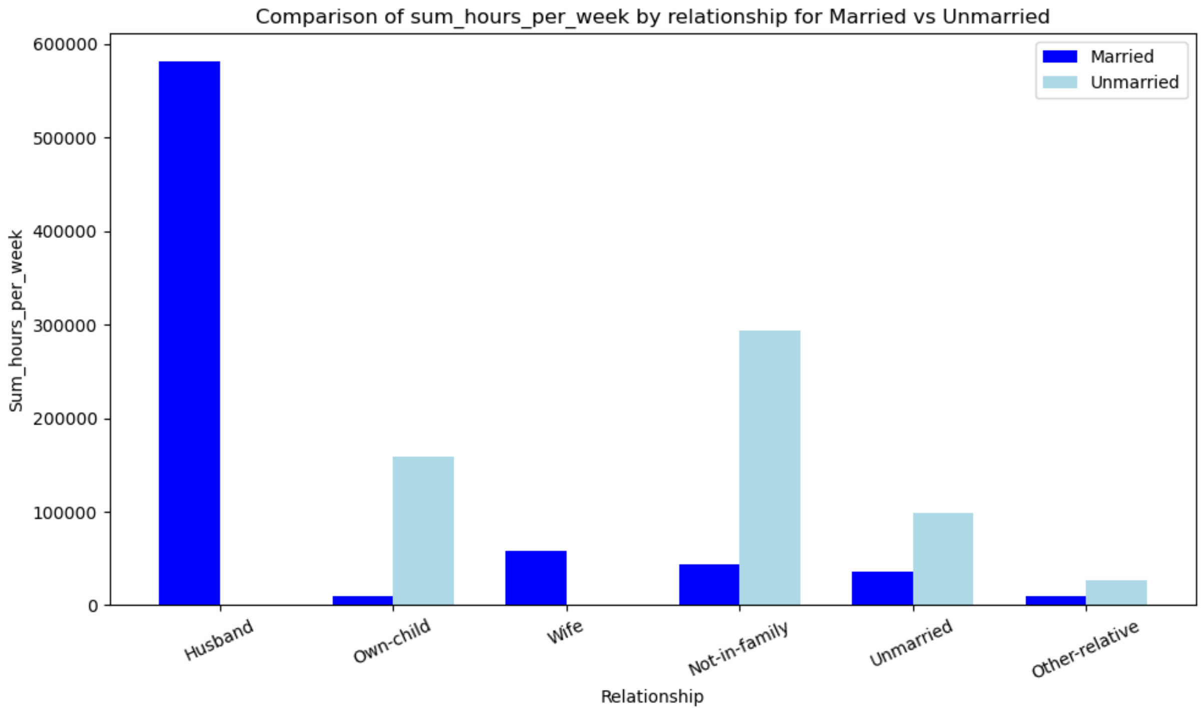

Figure 3: Sum of hours per week by relationship.

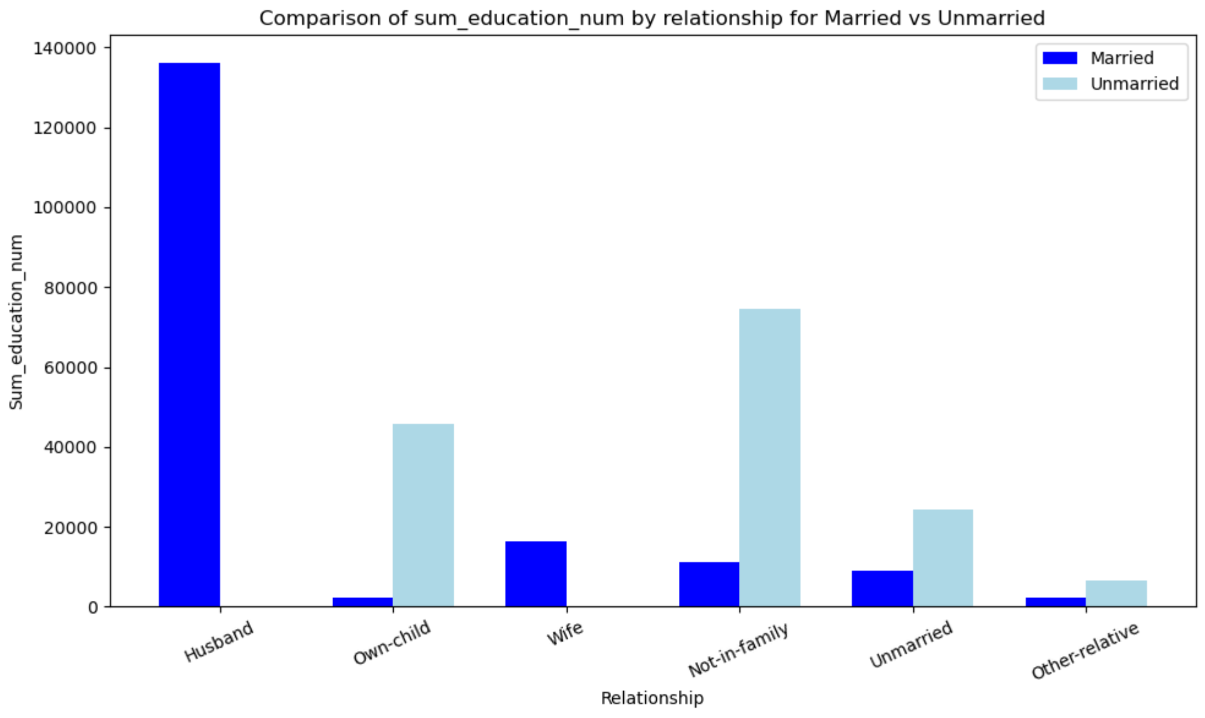

Figure 4: Sum of hours per week by relationship.

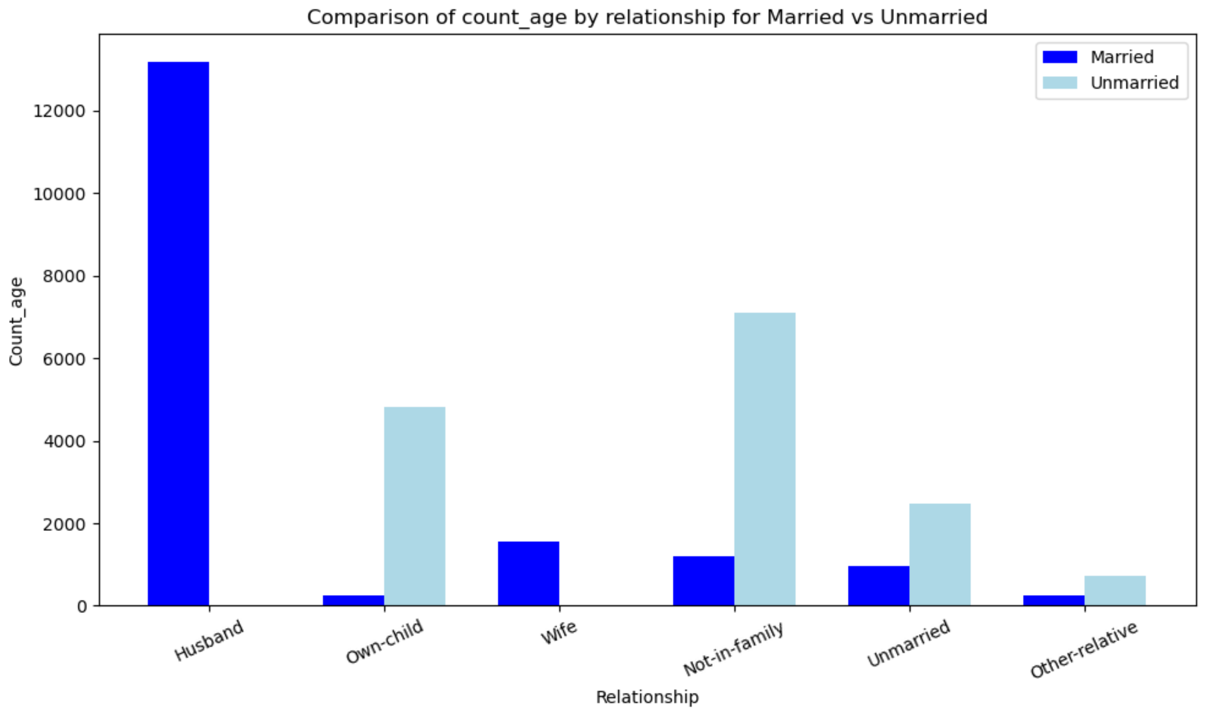

Figure 5: Sum of hours per week by relationship.

4.2 Performance

When using a local Postgres instance, we observed the performance of recommending the top-5 most interesting views from Census Income dataset (Kohavi, 1996) as Table 3 and Figure 6. In addition, we compared the result differences among three approaches (baseline, sharing-based optimizations, and pruning-based optimizations) and found out that all approaches came up the same recommendation views with very little error on utility scores, as shown in Table 4 and Figure 7. The performance result might slightly vary from run to run and could depend on multiple factors such as physical latency, CPU performance, and RAM size.

Method	Running Duration (seconds)
Baseline	3.06
Sharing-based	0.57
Pruning-based	0.66

Table 3: Performance (running duration) comparison across different approaches.

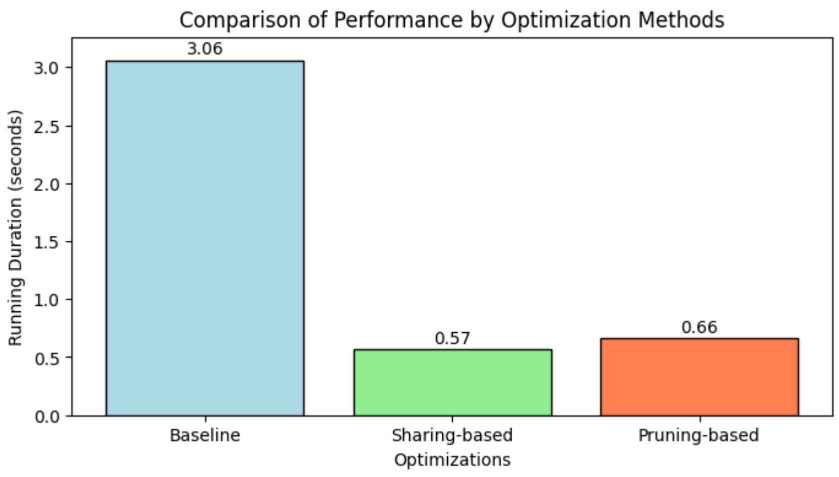

Figure 6: Performance (running duration) comparison across different approaches.

Attribute	Measurement	Baseline	Sharing-based	Pruning-based
Relationship	Capital Gain	21.23	21.30	21.24
Relationship	Capital Loss	20.44	20.62	20.44
Relationship	Hours per Week	19.31	19.45	19.31
Relationship	Education Number	19.16	19.33	19.16
Relationship	Age	18.81	19.01	18.81

Table 4: Result alignment across different approaches.

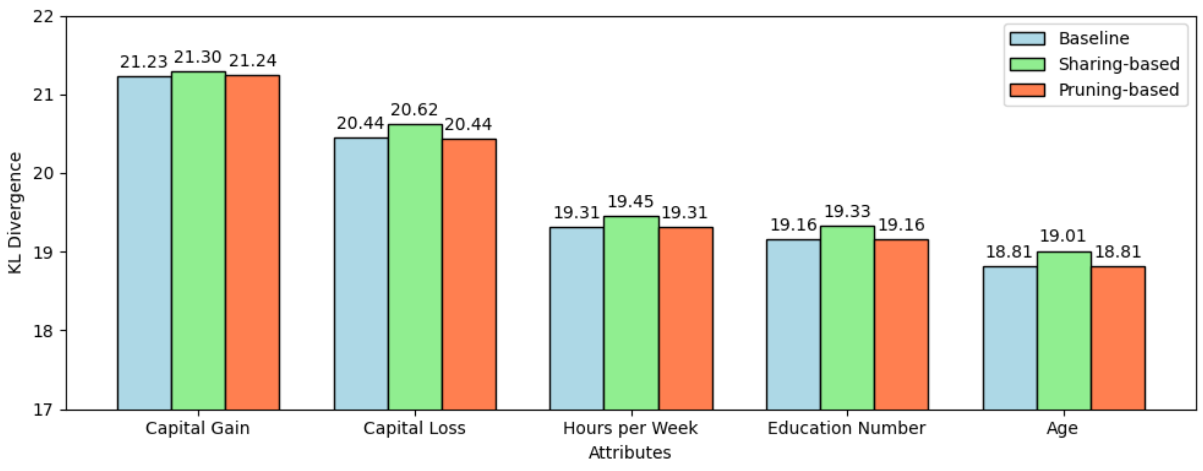

Figure 7: Result alignment across different approaches.

References

Ron Kohavi. 1996. Census Income. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5GP7S.

Robert Serfling. 1974. Probability inequalities for the sum in sampling without replacement. *Annals of Statistics*, 2:39–48.

Manasi Vartak, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. *SEEDB: Supporting Visual Analytics with Data-Driven Recommendations*. Technical Report.

Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. 2015. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 8:2182 – 2193.