LakeVisage: Towards Scalable, Flexible and Interactive Visualization Recommendation for Data Discovery over Data Lakes

Yihao Hu
Duke University
United States
yihao.hu@duke.edu

Jin Wang
Megagon Labs
United States
jin@megagon.ai

Sajjadur Rahman
Adobe
United States
sajjadurr@adobe.com

ABSTRACT

Data discovery from data lakes is an essential application in modern data science. While many previous studies focused on improving the efficiency and effectiveness of data discovery, little attention has been paid to the usability of such applications. In particular, exploring data discovery results can be cumbersome due to the cognitive load involved in understanding raw tabular results and identifying insights to draw conclusions. To address this challenge, we introduce a new problem: visualization recommendation for data discovery over data lakes, which aims to automatically identify visualizations that highlight relevant or desired trends in the results returned by data discovery engines. We propose LakeVisage, an end-to-end framework as the first solution to this problem. Given a data lake, a data discovery engine, and a user-specified query table, LakeVisage intelligently explores the space of visualizations and recommends the most useful and "interesting" visualization plans. To this end, we developed (i) approaches to smartly construct the candidate visualization plans from the results of the data discovery engine and (ii) effective pruning strategies to filter out less interesting plans so as to accelerate the visual analysis. Experimental results on real data lakes demonstrate that our proposed techniques can achieve an order-of-magnitude speedup in visualization recommendation. We also conduct a comprehensive user study to demonstrate that LakeVisage offers convenience to users in real data analysis applications by enabling them seamlessly get started with the tasks and performing explorations flexibly.

PVLDB Reference Format:
Yihao Hu, Jin Wang, and Sajjadur Rahman. LakeVisage: Towards Scalable, Flexible and Interactive Visualization Recommendation for Data Discovery over Data Lakes. PVLDB, 18(10): 3545 - 3558, 2025.
doi:10.14778/3748191.3748214

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yihaoh/datalake-vis.

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748214

This work was done when Yihao and Sajjadur were in Megagon Labs. Jin Wang is the corresponding author.

1 INTRODUCTION

Over the past few decades, there has been a significant growth in the number of open and shared datasets from governments, academic institutions, and enterprises. These massive collections of datasets, known as *data lakes*, open up new opportunities for innovation, economic growth, and social benefits [13, 32]. *Data discovery*, which aims to help users find and access specific data they need, is an essential operation for integrating and analyzing datasets from data lakes that are useful for various downstream tasks. Data discovery has become an important topic in the data management community with a specific focus on topics such as data exploration [15, 16, 43], table union search [14, 24, 36], joinable table discovery [9, 11, 58], and domain discovery [37, 59].

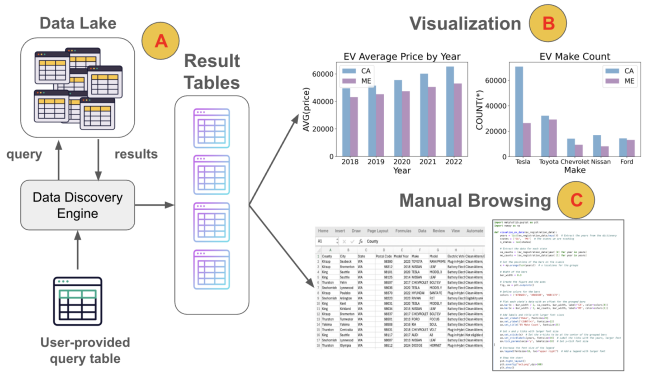

Figure 1: Motivation Example: analyzing EV sales trends from a data lake. (A) Given a query table, a data discovery engine returns top-k relevant result tables. (B) Visualizations recommended by LakeVisage **help an analyst quickly discover the marketing potential of economical EV brands in Maine. (C) Manual inspection in computational notebooks puts the onus on the analyst to reach the same conclusion.**

While many previous studies focus on improving the efficiency and effectiveness of data discovery, little attention is paid to its usability. Since the objective of data discovery is to provide richer data to boost data analysis tasks, it is essential to make the discovery results easily accessible and usable. In practical scenarios, the data discovery outcomes are always collections of tables with heterogeneous schema, wherein a subset of columns or rows of different tables are related [5, 17, 29, 46, 54]. These relations often

represent semantic relatedness obtained via approximations. Existing studies on data discovery place the onus on the user to analyze and make sense of such results, as the raw tables are simply returned for a given query. Therefore, exploring a collection of tables returned as data discovery results can be even more challenging. Although there have been some efforts in easing the exploration of data lakes [15, 35, 38], they target improving the usability of browsing the whole data lake instead of the data discovery results. Therefore, they cannot be utilized to resolve the above challenges. To address this issue, a natural approach is to provide visualizations, which is often the first step in data analysis [10, 28, 49, 52], as affordances to interactively explore data discovery results. With the help of visualization, it would be much easier for users to analyze the data discovery results. A typical example is shown in Figure 1:

Example 1.1. Suppose Aluko, a data scientist working for an EV company, has a repository of data about national electric vehicle registration and wants to gain more insights for potential marketing opportunities. To this end, she begins with a small table of sample data and identifies a collection of relevant result tables using data discovery engines, which are then further analyzed for business insights. Suppose the schema of the query table includes columns such as location, date, model, make, and retail price, among others, and each result table has a similar but not identical schema. While it is natural to first explore the average price of EVs for different states to identify potential markets, this would require manual efforts to collect metadata and write programs to perform the designated analysis task. As depicted in Figure 1C, when relying on manual browsing, it can potentially require significant coding and inspection efforts, especially when the data size is huge. Moreover, the iterative nature of exploratory analysis may require Aluko to repeat the above workflow several times before deriving insights, making it a cumbersome experience.

Meanwhile, LakeVisage alleviates the burden via taking all tables returned by the search engine, strategically resolving column alignments, and recommending interesting visualizations to Aluko as shown in Figure 1B. The first visualization shows that the average price of electric vehicles (EVs) in Maine is significantly lower than that in California, and the rate of price growth is also slower. The second visualization explains one potential reason: Toyota, an economic EV, has a significant market share in Maine, whereas Tesla, a high-end EV, dominates in California. Based on this observation, instead of spending a long time preparing data for analysis, Aluko can quickly conclude that Maine and California are suitable target markets for economic and luxury electric vehicles, respectively.

Many previous works provide scalable and interactive visualization recommendations for structured queries over relational data [21, 25, 28, 47, 49]. Given a structured query such as SQL, they can automatically identify the most interesting visualizations for analytical tasks. However, it is non-trivial to extend these techniques to the scenario of data discovery tasks over data lakes due to the following reasons: Firstly, since data discovery tasks aim at finding related tables from the data lake, it is essential to reflect the relationship between the query and result tables in the visualization. In addition, visualizations of multiple related columns among the data discovery results can introduce *perceptual scalability* challenges for the users, which exists when analyzing even a single table, i.e., viewing and exploring the information in such tables puts the cognitive burden on users [10, 41, 55]. Unlike previous work for relational databases, the visualization needs to present information from multiple tables in data discovery scenarios. Moreover, while the results of SQL queries are always a set of tuples, those of data discovery tasks are collections of tables. As a result, the search space of the visualization recommendation for data lakes would be much larger than that of relational databases.

In this paper, we study how visualization recommendation can help ease the exploration of the data discovery results from data lakes. To this end, we propose an end-to-end framework (LakeVisage[1]) that recommends visualizations as affordances to assist users in seamlessly exploring the results returned by a data discovery engine. Since there is no previous study on this problem, we first come up with a formal definition for it to illustrate how to build visualization over results across multiple tables. To illustrate the relatedness between tables in the results and the given query, we need to allocate result tables of data discovery into a certain number of *series* in the visualization, which could also help improve perceptual scalability. We provide a data-driven solution to efficiently generate high-quality results based on the data distribution of involved columns and develop effective pruning techniques to reduce the cost of handling unpromising visualization plans and accelerate the overall query processing time. These techniques could result in up to 30X of performance gain in total. In addition, we demonstrate the reliability of LakeVisage by conducting a comprehensive user study and showing that LakeVisage could result in high accuracy and shorter answer time for data science tasks compared with a literate programming tool without visualizations.

Our contributions in this paper are summarized as follows:

- We study the new research problem of visualization recommendation for data discovery over data lakes, providing the first formal definition.
- We design and implement LakeVisage, an end-to-end framework, as the solution to the problem and use table union search as a use case to illustrate our proposed techniques. Specifically, we develop novel techniques to construct visualizations over multiple result tables and propose a suite of pruning techniques to reduce execution time.
- We conduct comprehensive experiments over three public datasets for data discovery, demonstrating that our optimizations reduce execution time by an order of magnitude.
- We present the results of a systematic user study evaluating the usability and reliability of LakeVisage compared to Jupyter Notebooks, a widely used literate programming tool for exploring data discovery results among practitioners. The outcome of the user study further justifies that LakeVisage significantly eases the data analysis of data lake-related tasks.

The rest of this paper is organized as follows: Section 2 introduces the necessary background knowledge and formally defines the problem. Section 3 presents our proposed framework with technical details. Section 4 discusses some essential topics regarding

[1]LakeVisage is a portmanteau of the words lake (from data lakes) and envisage, thereby representing the goal of helping users comprehend data discovery results visually.

our proposed framework's flexibility and potential extensions. Section 5 illustrates the experimental results over real benchmarking datasets. Section 6 introduces the design and results of the user study. Section 7 surveys the related work, and Section 8 contains concluding remarks.

2 PRELIMINARY

2.1 Data Discovery

The core task of data discovery is to find related tables [46] in data lakes for a given query table. There are many definitions of table relatedness, and in this paper, we use Table Union Search [36] to illustrate our proposed techniques, which can be easily extended to support other data discovery tasks. Suppose the data lake is a collection of tables $\mathcal{T}$, and each table $T \in \mathcal{T}$ consists of one or more columns. To determine if two tables T_1, T_2 have enough similar content to be "unioned" together, a table union search engine generally computes a *table unionability score* U in two steps: (1) compute a *column unionability score* for each of all pairs of columns t_i, t_j where $t_i \in T_1, t_j \in T_2$ as $F(M(t_i), M(t_j))$, where F is a similarity score function and M is a column encoder [3, 12, 14, 24, 36], (2) use an aggregate mechanism A to compute the table unionability score by considering all previously computed column unionability scores. For a simple example, for any two tables with textual data in all columns, M can be the bag of tokens in a column, F can be the Jaccard similarity between the two bags, and A can be the sum of all column unionability scores. A general definition of the Table Union Search problem is then the following:

Definition 2.1 (Table Union Search). Given a collection of data lake tables $\mathcal{T}$ and a query table S, top-k table union search aims at finding a subset $\mathcal{S} \subseteq \mathcal{T}$ where $|\mathcal{S}| = k$ and $\forall T \in \mathcal{S}$ and $T' \in \mathcal{T} - \mathcal{S}$, we have $U(S, T) \geq U(S, T')$.

2.2 Visualization Recommendation

We now introduce the essential terminologies of visualization recommendation. A representative work in this field is SeeDB [49], which focuses on visualization recommendations for a relational database D with a snowflake schema. In visualization recommendation literatures, the setting is assumed to be database D with a snowflake schema. Within such a setting, there are three key aspects for visualization generation: *dimension attributes* $\mathcal{A}$ are the attributes to group-by in a visualization; *measure attributes* $\mathcal{M}$ are the attributes to perform aggregate on in the visualizations; and aggregate functions $\mathcal{F}$ (such as SUM, COUNT, and AVG) over measure attributes. Therefore, a *visualization plan* P is a function represented by a triple $\langle A, M, F \rangle$, where $A \in \mathcal{A}$, $M \in \mathcal{M}$, $F \in \mathcal{F}$. Intuitively, P can be regarded as a two-column table $\langle A, F(M) \rangle$, which can be displayed via standard visualization mechanisms, such as bar charts or trend lines. The visualization recommendation approaches typically focus on such visualization plans due to their reported popularity among users of visualization tools [34, 49]. Given a user query Q that aims to explore a subset of table D and results in a visualization P_Q, the goal of visualization recommendation is to identify other visualizations that are interesting with respect to P_Q on a predefined utility function. Examples of interestingness include larger deviation from the distribution underlying P_Q, which can be measured via utility functions such as Earth Mover's Distance (EMD), Euclidean Distance, Kullback-Leibler Divergence (K-L divergence), and Jenson-Shannon Distance [27, 49].

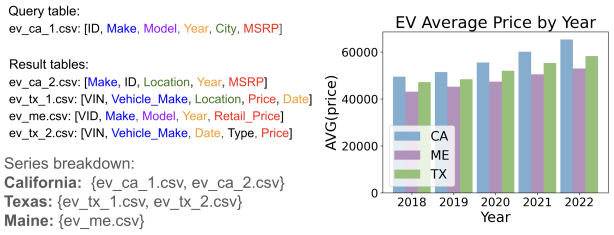

Figure 2: Illustration of Terminologies for Visualization

2.3 Problem Definition

Next, we will explain the formal definition of the problem visualization recommendation for data discovery using the aforementioned terminologies. Let Q denote the query table and $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ denote the set of k result tables of the Table Union Search problem, respectively. Suppose Q has u columns where the i^{th} column is denoted as q_i, and the x^{th} column of the y^{th} result table S_y is denoted as s_{xy}. Based on the definition of data discovery, each column $q \in Q$ is aligned with a set of columns in the result tables denoted as $C(q)$. For each $C(q)$, we have $|C(q)| \leq k$ since each column $q \in Q$ will align with at most one column in each result table $S \in \mathcal{S}$.

Recall that in the example introduced in Figure 1, the system can identify important visualizations for data science tasks by summarizing the information from multiple tables in the results of data discovery. To this end, we need to revise the definition of a visualization plan. We still follow the idea of related work introduced earlier by denoting a visualization plan using the triplet $P = \langle A, M, F \rangle$ to define visualization plans. While the visualization plans for relational databases in previous studies introduced above are two-column tables $\langle A, F(M) \rangle$, in the scenario of data discovery, the visualization needs to display results from multiple tables, specifically k. Additionally, since data discovery tasks aim at identifying tables relevant to the given query, it is essential to illustrate such relatedness in visualization plans. To satisfy this requirement, we propose the new concept of *series* that can be formally described as follows: given a query column q and the set of its aligned columns $C(q)$, a *series* γ is the ordered vertical concatenation of w aligned columns that are related with each other in $C(q)$ where $w \in [1, |C(q)|]$. We denote the set of series corresponding to $C(q)$ is as $\Gamma(q)$, where $1 < |\Gamma(q)| \leq |C(q)|$. And the set of all series for the query table Q is denoted as $\Gamma_Q = \cup_{q \in Q} \Gamma(q)$. In the visualization plan for data discovery, the bars under a value of dimension attribute A should correspond to a series $\gamma \in \Gamma_Q$, while the values of A should come from the union of all columns in $C(q)$ and $q \in Q$. In this way, we also improve perceptual scalability, as the number of series is no larger than that of aligned columns in the result tables. The definitions of the measurement attribute M and aggregate F are similar to previous work, and $F(M)$ returns a single real number. Here, we assume that F includes only one aggregation operation on one column for ease of presentation.

Example 2.2. We show a running example to illustrate the above terminologies. In the upper left part of Figure 2, it shows an example of a query table and its result tables, where the aligned columns are in the same color. For example, for the query column q of *MSRP* (i.e., manufacturer-suggested retail price), the set of aligned columns $C(q)$ is $\{MSRP, Price, Retail\_Price, Price\}$. The right part of Figure 2 presents an expanded version of the first visualization shown in Figure 1, and the series "Texas" is added as more result tables are returned (i.e., *ev_tx_1* and *ev_tx_2*). Here the dimension attribute A is *Year* and there are 5 values. The set of series for above $C(q)$ is $\Gamma = \{ev\_ca, ev\_tx, ev\_me\}$ as shown in bottom left of the figure, where *CA* is the union of column *MSRP* from table *ev_ca_1* and *ev_ca_2*. Similarly, *TX* is the union of column *price* from both *ev_tx_1* and *ev_tx_2*, and *ME* is the column *Retail_Price* from table *ev_me*. The measure attribute M is the count of all rows (or equivalently, *ID*) and the aggregate F is COUNT.

Following the practice of previous studies [27, 49], we also use Earth Mover's Distance (EMD) as the utility function $\mathcal{D}$ to evaluate whether a visualization plan is interesting. Note that, in keeping with the classical visualization recommendation literature, the focus of this work is to provide a framework that facilitates the integration of any suitable metric, rather than finding the best utility metric. Suppose there are v series in Γ, the *utility score* of a visualization plan P can be calculated with Equation (1):

$$\mathcal{D}(P) = \frac{2}{v * (v-1)} \sum_{i,j \in [1,v], i \neq j} EMD(\gamma_i, \gamma_j) \tag{1}$$

Here we use γ_i interchangeably to denote the associated numerical vector of a series; the vector dimension equals the number of unique values in A, and the value in each dimension is the result of $F(M)$ for each value in A. Finally, we have our formal problem definition of visualization recommendation for data discovery as Definition 2.3:

Definition 2.3. Given a query table Q and its set of result tables $\mathcal{S}$, the visualization recommendation for data discovery problems aims at finding top-n most interesting visualization plans $\mathcal{P}$ where $\forall P \in \mathcal{P}, P' \notin \mathcal{P}$ we have $\mathcal{D}(P) \geq \mathcal{D}(P')$

3 METHODOLOGY

3.1 System Overview

We first introduce the front-end of LakeVisage as shown in Figure 3. It provides an interactive user interface that allows users to specify the visualizations and returns the top-ranked visualization plans recommended by our proposed algorithms. The web-based front end consists of five components: (A) a query builder where the user specifies the query table and path to the data lake storage; (B) a schema viewer which displays the schema information of query and result tables; (C) a recommendations panel which displays the recommended visualization plans, (D) a detailed view which highlights the plan a user selected from the recommendations; and (E) a plan builder for users to specify customized visualization plans.

LakeVisage supports an end-to-end pipeline that can be built on top of any data discovery engine. The overall workflow is shown in Figure 4. Given the input table, the data discovery framework first retrieves the top-k relevant result tables from the data lake. Then there are three major steps for the visualization recommendation process: *Series Creation*, *Candidate Generation* and *Results Ranking*. The Series Creation step creates the series that serves as the bars denoting the $F(M)$ values in the visualization chart for each dimension attribute value. The Candidate Generation step identifies promising visualization plans from the large search space, reducing the computational overhead of evaluating utility scores. The Results Ranking step verifies the utility score for all candidate plans and returns the top-n ones as the recommendation results.

One important issue to be addressed in data lake scenarios is handling heterogeneous data formats. LakeVisage supports three data types: categorical, numerical, and textual. Based on the definition of visualization plans, we only consider categorical and numerical columns as the candidates for measuring attributes (M) in a plan. When a column appears as a dimension attribute (A), we transform its entries, i.e., cells in the table column, to categorical values when they are not. We discretize the column values into several bins for numerical columns and map each entry to the respective bins. Each bin corresponds to a value of the dimension attribute. For textual columns, we transform each entry into a low-dimensional embedding vector. Then we perform a clustering over the embedding of all cells and treat each cluster as a dimension attribute value. Similar discretization approaches have been employed by tools for interactive analytics on textual data [42, 56].

3.2 Series Creation for Visualization

In order to satisfy the requirements of showing the relatedness between tables in data discovery results, we need to construct high-quality series collection $\Gamma(q)$ for each $q \in Q$ from its aligned columns $C(q)$ introduced before to serve as the bars for each attribute dimension value in the visualization chart. Three general principles guides the series creation: (i) the data semantics of the columns belonging to the same series should be similar; (ii) the overall utility score of the series over different attribute dimension values should be large to make more contributions to the interestingness of the plan; and (iii) there should not be too many series to ensure the perceptual scalability.

Among the above, the Series Creation step takes care of the first principle while the Result Ranking step covers the second and the third. In the simplest scenario, each column $s_{xy} \in C(q)$ is regarded as a series. However, such a case may lead to a loss of context regarding the relatedness of columns. Moreover, as the number of series increases, users are overloaded with information, leading to perceptual scalability challenges. To acquire high-quality series with relevant columns, we need to define a mechanism for evaluating their relatedness. We start from a heuristic method based on syntactic similarity as the baseline: we consider columns with similarity scores larger than a pre-defined threshold as related ones. Specifically, we consider the criteria of columns with different data formats as follows:

- Categorical: Regard two columns as two sets and consider the syntactic similarity between them.
- Numerical: Identify the range of two columns and consider their overlaps.
- Textual: Regard two columns as two bags of words and consider syntactic similarity between them.

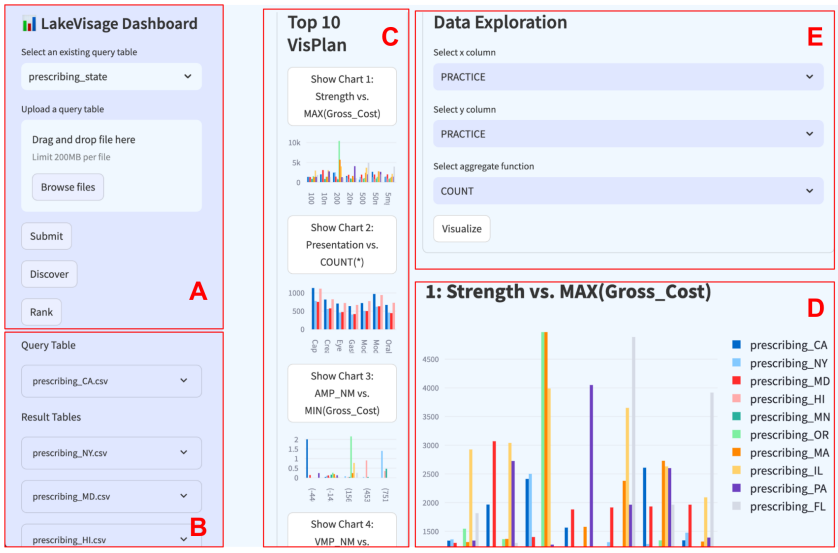

Figure 3: The Front-end of LakeVisage.

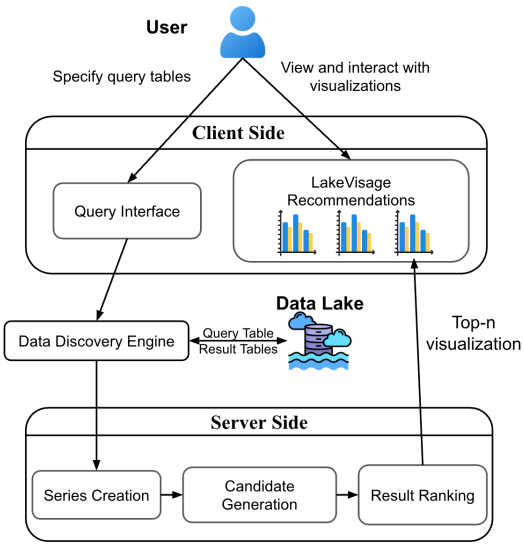

Figure 4: The Overall Architecture.

We use Jaccard similarity as a representative of syntactic similarity metrics in our current implementation and experiments since different metrics tend to have similar results in this step.

Nevertheless, the above syntactic similarity may not be an ideal solution due to two concerns: on one hand, the simple similarity scores fail to convey enough semantic information across multiple columns within a series. On the other hand, traversing two columns is required to compute the similarity score. Thus, the time complexity is related to the size of tables the columns belong to, which is expensive considering the large size of data lake tables. To address these issues, we introduce a data-driven approach utilizing statistical tools. The high-level idea is to treat each column as a random variable with values for each entry drawn from a distribution. Then we can determine whether two columns should be grouped together based on their distributions. Given two columns [2] If their distributions are close enough, they will be considered as belonging to one series and thus merged into one series. To reach this goal, we employ **Pearson's chi-square test** [18], which measures whether the observed frequency distribution of a random variable is significantly different from its expected frequency distribution. The statistic is calculated by Equation (2):

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i} \tag{2}$$

where N is the number of categories, O_i and E_i is the observed and expected frequency of each category.

In our scenario, we can regard the two columns as the observed and expected distributions, respectively, and apply this tool. For the column regarded as observed one, we conduct random sampling to obtain W samples to serve as the observations O_i; For the column regarded as the expected one, we need to estimate its statistics to decide the above value of E_i. We assume that the expected column follows the exponential distribution family [3], which covers most common distributions such as Bernoulli, normal, and gamma distributions. The statistics can then be estimated by randomly selecting W samples from the column and then conducting Maximum Likelihood Estimation. We empirically observed that setting W to 500, which is less than 1% of most tables, yielded very promising results. After obtaining values of O_i and E_i using the above sampling and estimation approach, we can then calculate the χ^2 test statistic and decide whether to reject the null hypothesis, i.e., the two data distributions are similar, accordingly. Note that the overhead of estimating E_i values would be trivial. Once we decide which columns belong to the same series, we can decide the statistics of the newly formed series based on those of the two columns. For instance, suppose two columns c_1 and c_2 follow Gaussian distributions $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$, respectively, then the series consists of them will follow the Gaussian distributions $N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Thus, the required number of estimations for a given column $q \in Q$ is no more than the cardinality of its aligned columns $|C(q)|$.

Another issue to be resolved in series creation is the large search space. Given the column $q \in Q$, the number of potential candidates for series collection is $O(2^n)$, where n is the number of aligned columns in result tables ($|C(q)|$). To reduce computational overhead, we propose an iterative process for finding promising series collections without traversing the entire search space. The high-level idea is that we sort all columns in $C(q)$ in ascending order of cardinality and initialize each series with one single column. Then, we start from the smallest columns and conduct the above statistical tests on a pair of columns. If they pass the test, they will be merged into one new series and proceed to the next round. In each round, we will only merge one series pair. The whole process will end if there are only two series left or no pair of series can be further merged. This process can be efficiently implemented with

[2]The comparison can also happen between two series where each series is a combination of multiple columns. We will only mention columns when introducing the techniques for the ease of presentation.

[3]https://en.wikipedia.org/wiki/Exponential_family

Algorithm 1: Series Creation

Input: The Query table Q; The set of result tables $\mathcal{S}$
Output: The Series Collection Γ_Q

```
Initialize Γ_Q = ∅;
foreach q ∈ Q do
    Initial Γ(q) = ∅;
    Obtain C(q) from result tables S;
    Sort C(q) in ascending order of cardinality;
    for c_j ∈ C(q) do
        Γ(q) ← Γ(q) ∪ c_j;
    Initialize cursor j = 0;
    while |Γ(q)| > 2 and j < |Γ(q)| do
        Conduct statistic test on Γ(q)[j] and Γ(q)[j + 1];
        if Test pass then
            Merge them into a new series, update Γ(q);
            Reset j = 0;
        else
            j+ = 1;
    Γ_Q ← Γ_Q ∪ Γ(q);
return Γ_Q;
```

data structures like Union Find. Based on the above discussion, we propose the solution for series creation as shown in Algorithm 1.

Based on the above discussion, we propose the solution for series creation as shown in Algorithm 1. The complexity is analyzed as follows: First, $\forall q \in Q$ the cardinality of $|C(q)|$ is $O(k)$ based on the problem setting where k is the number of result tables. The time to sort each $C(q)$ is $O(k \log k)$. Given two columns in $C(q)$, the time to perform chi-square test is $O(W)$ where W is the sample size; and the number of test is $O(k)$. Thus the time complexity for handling each $C(q)$ is $O(k \log k + k * W)$, while the overall time complexity of the u columns in the query table would be $O(u * k * (\log k + W))$. The overall computation overhead would be trivial since W and k will be relatively small numbers.

3.3 Efficient Candidate Generation

After creating the series for query and result tables, we can construct the visualization plans accordingly. As discussed before in Section 2.3, each plan is a triplet $P = \langle A, M, F \rangle$. Here, the potential candidate of dimension attributes A is all query columns $q \in Q$ since a column from result tables might not have aligned columns in the query table; Meanwhile, the measurement attributes M are all the categorical and numerical columns in the query and result tables. A valid visualization plan should satisfy two requirements: (i) There should be no overlap between dimension and measurement attributes, i.e., $A \neq M$ and $M \notin C(A)$; (ii) The result of grouping values in A by M should not be empty. For the aggregates F, we consider the common ones such as COUNT, MIN, MAX, AVG, and SUM, among others. Following this route, a straightforward solution is to enumerate all plans and compute each plan's utility score. Then, the result will be plans with top-n highest utility scores.

3.3.1 Avoid Redundant Computation. However, such a linear scan is rather expensive due to the overhead of group-by operations for each plan. Since a group-by operation is required to traverse the tables associated with columns in A and M, its cost is directly related to the table size, which can be very large in the data lake scenario. To address this issue, we need to avoid redundant computation in computing group-by. We observe that when grouping by a given dimension attribute A, the result of COUNT aggregation is the same for all measurement attributes M. Thus, when enumerating the plans, if the COUNT aggregation is already calculated for a previous plan group by A, we can share the results for all M. The results of AVG aggregation can be obtained from those of SUM and COUNT for the same A and M, requiring no redundant computation.

3.3.2 Bound Estimation with Sampling. We can optimize further by reducing the volume of data accessed during the candidate generation process. To reach this goal, we borrow the idea from previous works in Approximate Query Processing (AQP) [1]. The high-level idea of AQP is to first execute queries on a small, sampled subset of the entire dataset and obtain an initial answer quickly. Then, the size of the sampled dataset grows gradually, and the query results become increasingly accurate. The entire process will terminate when the budget for accessing data is exhausted or a certain accuracy guarantee is met. In our problem setting, when computing the group-by and utility scores, we can gradually examine a subset of the table instead of using the entire table for computation at once. In this process, the temporary utility score computed from a subset of data can also serve as a lower bound for the top-n results: if the utility score of a plan is already lower than this lower bound, the plan can be pruned without computing it over the full tables. To reach this goal, we employ the Hoeffding-Serfling inequality [51] from the domain of statistics to deduce such a bound, which is formally stated in Theorem 3.1:

Theorem 3.1. *Let $\mathcal{Y} = y_1, y_2, ..., y_N$ be a set of values in range $[0, 1]$ with a mean value μ; Let $Y_1, Y_2, ..., Y_m$ be a sequence of random variables drawn from $\mathcal{Y}$ without replacement. For every $k \in [1, N)$ and a probability $\delta > 0$:*

$$Pr[\max_{k \leq m < N} |\frac{1}{m}\sum_{i=1}^{m} Y_i - \mu| \geq \epsilon_m] \leq \delta \tag{3}$$

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

The core idea of this theorem is that the bias between results computed over data samples and the true result is within a given interval with size ϵ, which is related to the number of samples (m in the theorem). The more samples there are, the smaller the value ϵ will be, which means the results will be more accurate. Following the practice of hypothesis testing, the value of δ is set as 0.05. In our problem setting, each Y_i above is regarded as an estimate of the utility score for a given visualization plan P. Then the value of ϵ can be computed based on Theorem 3.1. Suppose the estimated score is D, then the lower (upper) bound would be $D - \epsilon$ ($D + \epsilon$).

3.3.3 The Pruning Algorithm. Although the high-level idea of Theorem 3.1 has been employed in previous studies about scalable data visualization [25, 30, 40, 49], their solutions cannot be directly applied to our problem. To realize the above idea in our problem setting, we propose an effective pruning strategy as shown in Algorithm 2. For each column in the query table, it first splits itself and all its associated tables into batches with random shuffles (line

1) and applies one batch in the computation at a time (line 4). Since the results between batches might change drastically, it restarts the ranking at the beginning of each batch by clearing the heap (line 5). To facilitate the pruning, a global lower bound is maintained for the top-n' plans in the heap (line 6). For each plan in $\mathcal{P}$, the new utility score is calculated by taking the average of all past scores plus the current one for this plan. It then computes the interval ϵ and deduces the lower and upper bounds in the plan. If the heap size has not yet reached n', it simply adds the plan and updates the global lower bound (lines 13-15). Otherwise, depending on whether the upper bound of P is smaller than the lower bound of the top n' in the heap, it decides to discard P (lines 17-18) or add it to the heap (line 20). If P has a score that gets it into the top n' in the heap, we need to consider updating the global lower bound (lines 16-22). Finally, after looping through all plans in $\mathcal{P}$, if there are exactly n' plans left in the heap, the process is terminated (lines 23-24).

Algorithm 2: Candidate Generation

Input: Query column q; Result tables $\mathcal{S}$; Number of results n'
Output: The set of candidate plans $\mathcal{H}$

1 Split q and tables $\mathcal{S}$ into several batches $\mathcal{B}$;
2 Identify the potential set of visualization plans $\mathcal{P}$;
3 Initialize $\mathcal{H} = \emptyset$;
4 **foreach** $B \in \mathcal{B}$ **do**
5 $\mathcal{H} = \emptyset$;
6 $L_{overall} \leftarrow$ the min lower bound of top n' plans in $\mathcal{H}$, initialize to inf;
7 **foreach** $P \in \mathcal{P}$ **do**
8 Compute $P.F$ over $P.A$ and $P.M$ over B, reuse the COUNT results when necessary;
9 $P.score \leftarrow \mathcal{D}(P_B)$; // $\mathcal{D}(P_B)$ is the partial results computed on B
10 Compute the interval size ϵ;
11 $P.L = \mathcal{D}(P_B) - \epsilon$;
12 $P.U = \mathcal{D}(P_B) + \epsilon$;
13 **if** $|\mathcal{H}| < n'$ **then**
14 Add P into $\mathcal{H}$;
15 $L_{overall} = min(L_{overall}, P.L)$;
16 **else**
17 **if** $P.U < L_{overall}$ **then**
18 $\mathcal{P}.remove(P)$;
19 **else**
20 Add P into $\mathcal{H}$;
21 **if** P *is in the top* n' **then**
22 $L_{overall} = min(L_{overall}, P.L)$;
23 **if** $|\mathcal{H}| = n'$ **then**
24 **return** $\mathcal{H}$;
25 **return** $\mathcal{H}$;

The pruning methods introduced in Algorithm 2 might involve false negatives because (i) the estimation is with a probability δ; (ii) the batch shuffling process involves randomness that might discard promising plans with low utility score in earlier batches. While trading quality for faster execution, we will later show in Section 5.2 that the proposed solution can achieve a reasonable trade-off between quality and execution time. Here, the hyperparameter n' decides the number of candidates to be verified. We set it as n empirically in our implementation. We will apply the above approach on all the u query columns in the query table and obtain $q * n'$ candidates in the candidate generation step. Finally, in the Result Ranking step, we will verify each candidate's true value and select the top-n highest results.

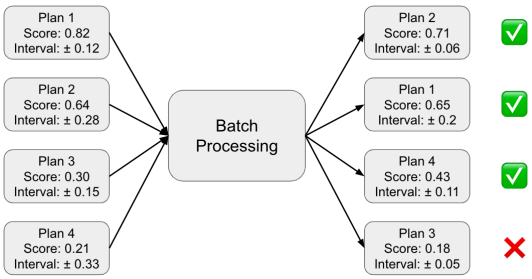

Figure 5: A Running Example of Pruning Process

Example 3.2. We provide a running example in Figure 5 to illustrate how Algorithm 2 works. Suppose there are 4 plans in the second batch ranked in descending order of estimated scores, and their scores and intervals are shown on the left side. After batch processing, their scores and intervals are updated, and their rankings are also updated accordingly. If we would like to look for the top 2 plans, then plans 2 and 1 are under consideration. Plan 4 is still kept because its upper bound (0.54) overlaps with the lower bound of plan 1 (0.45). However, Plan 3 can be pruned because its upper bound of 0.23 does not overlap with that of Plan 1.

4 DISCUSSION

In this section, we discuss the potential opportunities of expanding the functionalities of our proposed framework.

We begin with the discussion about limitations of our work. Firstly, as our main contribution lies in the definition and proposal of a general framework for implementing visualization recommendation in data discovery, the current version of LakeVisage may not perform well under certain circumstances or real world data lakes in larger scales. However, given specific assumptions, one can always easily adjust some of the building blocks of LakeVisage to suit their need. For example, the utility metric EMD can be replaced with Kurtosis[4] [39] if users specifically look for non-Gaussian or Gaussian distributions instead of the differences between distributions. Due to the inherent characteristics of a data lake, such as the lack of high-quality metadata and inconsistent schema, it is possible for LakeVisage to produce false positives (i.e., merging two irrelevant columns into one series). Should additional metadata or external knowledge be provided, LakeVisage would generate accurate series with much fewer errors. For instance, if we know that each result table pertains to the records of a city for a given year, we can group tables about the same city into a single series. Then the visualization will present the results of each city from

[4]A metric that measures how "Gaussian" a distribution is.

multiple years, aggregated from several result tables. Our proposed data-driven approach can serve as a reasonable solution in general scenarios. In summary, LakeVisage is flexible and easy to integrate user-defined approaches to accommodate different assumptions.

Finally, although the current LakeVisage only supports Table Union Search as the representative of data discovery task, it can be extended to other column-align-based tasks. We would like to illustrate this point with a discussion of the Joinable Table Discovery task [9, 58, 59]. Given a query table and a collection of tables, it aims to find all tables that can be joined with the query table. The output of this task is similar to table union search, i.e., a set of result tables where the potential joinable columns are highlighted in the search result. Thus LakeVisage can be directly applied to support joinable table discovery. One additional requirement of this task is to provide some insights for the pairs of joinable columns. We can build a hierarchical visualization mechanism for columns and tables following the practice of previous work [41]. For tasks based on row alignment, we would need to define new mechanisms of series creation and candidate pruning.

5 EVALUATION

5.1 Experiment Setup

Table 1: The statistics of datasets

Dataset	# Query Table	# Data Lake Table	Avg # Row per Table	Size (GB)
TUS	100	5,044	2,346	1.5
SANTOS	80	11,086	7,706	11
LakeBench	3,171	4,028	119,281	21

5.1.1 Datasets. We conducted experiments on three public datasets that have been widely evaluated in previous studies on data discovery. TUS [36] is the first benchmarking datasets for table union search. SANTOS [24] is created from Open Data and simulates the real data lake scenarios. LakeBench [7] is so far the largest benchmarking dataset for unionable and joinable table related tasks with rich manually created ground truth. Here we use its sub-task of OpenData Large. The detailed information can be found in Table 1. For each dataset, we ran each query with the Starmie [14] framework and obtained the top-ranked result tables. Note that our proposed techniques are not limited to a specific definition of unionability and could also be easily applied to other data discovery engines proposed in previous studies such as [24, 36].

5.1.2 Compared methods. Since there is no previous study on the problem of visualization recommendation for data discovery from data lakes, and as illustrated in Section 2, it is non-trivial to extend existing data visualization approaches to our problem. Here, we present the performance of each proposed technique in Section 3. Therefore, we include the following methods for comparison: NoMerge is the method that treats each column in the result table as a series, i.e., there is no series creation process. Overlap is the baseline method of series creation based on syntactic similarity introduced in Section 3.2; Stats is the data-driven approach for series creation introduced in Section 3.2; Prune is the method of applying the pruning techniques in Section 3.3 on the basis of Stats.

5.1.3 Evaluation metrics. We consider both the efficiency and effectiveness of each compared method introduced above. We use average execution time per query as the metric for efficiency. For effectiveness, we evaluated our approach based on the practice of a previous study [49]: we compute the average utility score of the top-n results for all queries in a dataset. The higher the average utility score, the more effective the compared method is. For the general usability of our solution, we will further illustrate in the user study later in Section 6.

5.1.4 Environment. We implemented all algorithms with Python. The experiments were run on a server with 1 AMD EPYC 7R32 48-core processor and 192GB RAM. We ran all experiments 5 times and reported the average performance. We will fix the number of returned visualization plans n as 10 by default.

5.2 Results

First, we report the results of the execution time of all proposed methods in Figure 6. In this experiment, we vary the number of result tables k in data discovery from 10 to 50 and observe the performance of each method. And we have the following observations: Firstly, Stats and Prune achieve better overall performance under all settings, which illustrates the necessity of techniques in the process of series creation and candidate generation. Specifically, Prune outperforms other compared methods by up to 30 times. For instance, in the LakeBench dataset when $k = 40$, the average execution time per query of Prune is 9.59 seconds, and while that for Stats, Overlap and NoMerge is 33.98, 277.09 and 95.64 seconds, respectively. Secondly, Overlap always performs the worst among all methods. The reason is that it takes a very long time to traverse the columns to compute the syntactic similarity, while Stats only need to estimate and check some statistics. Besides, the execution time of Overlap increases sharply along with the number of result tables, which is also due to the need to traverse columns. Thirdly, the gap in performance between different methods on TUS is relatively small, while that on the LakeBench is more obvious. As shown in the analysis in Section 3.3, this is due to the performance of algorithms being closely related to the average number of rows in the query and result tables. Since the tables in TUS are generally not so large as illustrated in Table 1, the bottleneck in efficiency is not as obvious as the other two datasets. Lastly, the execution time is relatively constant for all methods but Overlap. This is because NoMerge does not involve computation of series creation, while the complexity of Stats and Prune is independent of the total number of rows of all result tables.

Then we show the effectiveness of all proposed methods. As discussed in Section 3, it is not realistic to obtain the optimal overall utility score due to the exponential search space. Thus, instead of computing the recall or accuracy, we evaluate the quality of top-n results by looking at the average utility score of them. Since both Stats and Prune relied on statistical methods that involve randomness, we also include the error bars of these methods to denote the lower and upper bounds of them in multiple runs. The results are shown in Figure 7. We can see that, generally speaking, Stats has the overall best results in effectiveness, which achieves the highest scores under all settings. For example, on the SANTOS dataset when $k = 30$, the average utility score of NoMerge, Overlap, Stats, and Prune is 0.4, 0.3, 0.53 and 0.42 respectively. Note that since NoMerge always has a larger number of bars than Stats and

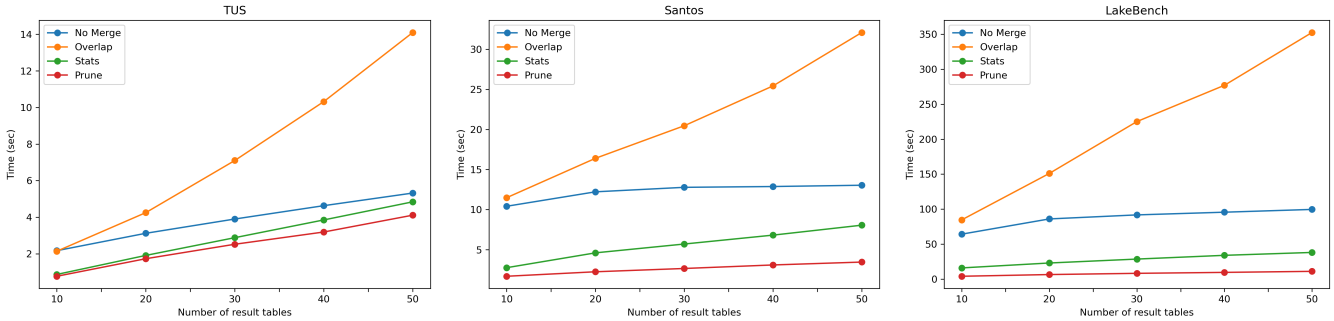

Figure 6: Results of Proposed Techniques: Execution Time

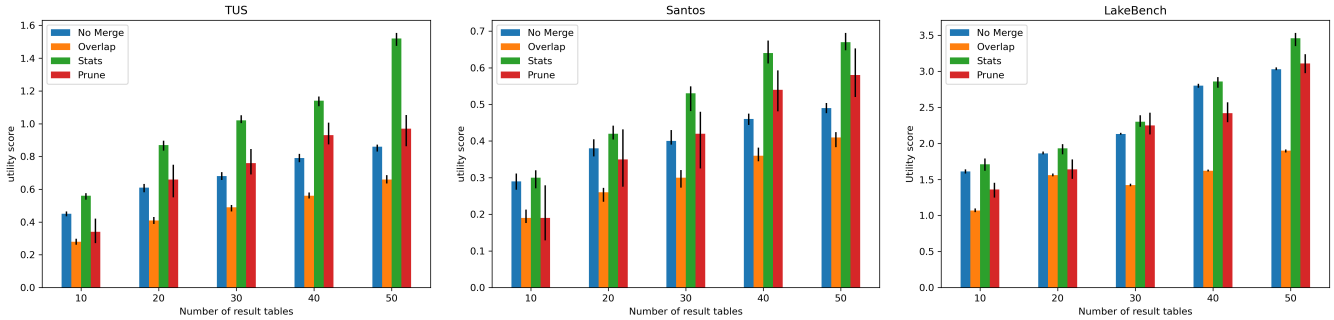

Figure 7: Results of Proposed Techniques: Effectiveness. Error bars indicate lower/upper bounds for methods with randomness.

Prune, the total utility score tends to be larger based on the way to calculate it as shown in Equation 1. The actual performance of Stats and Prune will be even more outstanding, considering the penalty regarding perceptual scalability brought by the number of bars in NoMerge. Here, we did not apply such penalties in the utility score to ensure consistency with the previous study [49]. The pruning strategies developed in Section 3.3 involved some false negatives that might discard some promising visualization plans. Therefore, we can consider Prune makes a trade-off between effectiveness and efficiency; thus its effectiveness results are worse than Stats. Besides, there is also an observation that the advantage of Stats becomes more obvious when there is a larger number of result tables. This is because Stats is a data-driven approach, and more data could result in more accurate estimation. Moreover, we also see that NoMerge performs generally better than Overlap since syntactically similar columns do not necessarily lead to related columns. It also shows that simple heuristics cannot easily handle the series creation step, and it is essential to introduce the data-driven approach, as we did.

Finally, we also investigate the scalability of our LakeVisage framework (i.e., above Prune method) w.r.t. the other two important factors: the data size of involved tables and the number of returned visualization plans. Since LakeBench is much larger than the other two datasets, we only conduct scalability experiments on it. To evaluate the scalability with respect to data scale, we select the top-20 queries with the largest number of rows in the query and result tables from LakeBench. For these queries, we vary the data scale

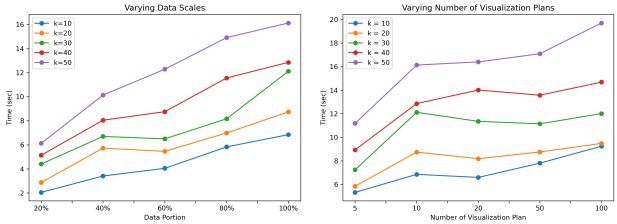

Figure 8: Results of Scalability on LakeBench dataset

from 20% to 100% for all the query and result tables and observe the performance of our proposed method. As shown in the left sub-figure of Figure 8, LakeVisage achieves near-linear scalability under all settings. Then, we vary the number of returned visualization plans, n, and show the results in the right sub-figure of Figure 8. We can see that the execution time stays stable before n reaches 50 and sometimes larger n even leads to less time. The reason is that when doing the group-by operations to generate candidates, many plans can have the same dimension attributes but different measurement attributes; thus, they have the same utility score. When the number of n increases, the utility score of the top element on the min heap might not change, and our search algorithm converges at a similar time or even faster due to the shared computation of aggregation developed in Section 3.3. When n is larger than 50, we observe that the utility score of the top element also increases greatly, and thus, it takes more time to finish searching.

5.3 Case Study

Next we show two case studies about some interesting visualization plans recommended by our proposed system. Our goal is to illustrate the useful insights gained from data discovery through the visualizations provided by the proposed techniques. First, we would like to illustrate an example of series creation in addition to the data discovery results. Figure 9 shows an example with a query table and result tables about cars. This example is about a car sales scenario that presents information such as the price, make, and date of the cars. In this visualization, the dimension attribute A is a numerical column *Year* which is divided into several bins as introduced in Section 3.1; the measure attribute M is a numerical column *Price* and the aggregate F is AVG. Specifically, the series is created based on the data distribution of the *Year* column in the query and result tables. We find that the created series can illustrate a correlation with the fuel of cars [5]. The visualization of the above series also reveals that the number of Hybrid/Electric cars has grown much faster than that of Gas cars over the last 10 years. It would save data scientists from the enormous efforts of manual programming to discover such a fact.

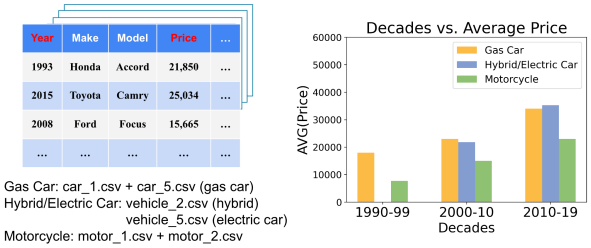

Figure 9: Case Study:the series created based on distribution of Year shows an interesting trend about fuel of cars.

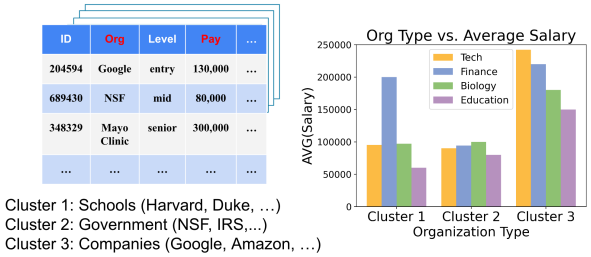

Figure 10: Case Study: A Visualization Plan with Textual Columns as Dimension Attribute.

Another important case we want to show is about the situation where textual columns serve as the dimensional attribute A in a visualization plan. As discussed previously, a key difference between LakeVisage and previous data visualization works for relational databases is that our work must handle textual columns and recognize visualization plans composed with them. Figure 10 shows an example with a query table about employee information, and result tables with similar contents. And LakeVisage created a visualization plan where A is a textual column Organization, M is Salary and F is AVG. Here, the Organization columns consist of the names of different organizations where employees work. Our approach splits the cells of such columns into several clusters, and each value of the dimensional attribute corresponds to a cluster, as shown in the bottom left of Figure 10. Here, we identify the latent semantics of the clustering results: each cluster corresponds to a specific type of organization, such as schools, governments, and companies. This visualization plan could further illustrate the average salary in different types of organizations, rather than a specific one. We can see that even the simple heuristics in Section 3.1 could help discover such interesting visualizations.

6 USER STUDY

6.1 Design

We compare LakeVisage with literate programming tools such as computational notebooks typically used by data scientists for exploring data discovery results in data lakes.

Datasets and quiz tasks. We selected two query tables from the popular Open Data repository [6] and retrieved the top-10 unionable tables returned by Starmie [14] as data discovery results. The two query tables were related to pet and medical domains which could represent real world tasks for data scientists. The number of columns in the two query tables is 11 and 12, respectively. Table 2 lists the questions provided to the participants when using computational notebooks. Meanwhile, to make a fair comparison, questions for the visualization systems are similar in difficulty but with different attributes and predicates. These questions were designed to introduce an increasing level of difficulty to the participants. Therefore, we did not randomize the order of the questions. These questions represent a range of data analysis tasks that users typically perform over data discovery results (see supplementary material for details.) We leverage the typology of data exploration tasks [4] to identify the purposes and actions required for each question. This typology encompasses a range of domain-independent tasks on visual data representations and has been widely utilized as a guideline for developing models of visualization systems and defining the scope of tasks in various domains, e.g., interactive task authoring, document mining, and multivariate network analysis [41]. All of these tasks required participants to complete several actions representing one of the following purposes: `browse` (searching based on characteristics where location is unknown), `lookup` (searching based on entities where location is known), `identify` (returning the characteristics of entity found during search), `generate/record` (generation or recording of new information), and `compare` (returning characteristics of multiple entities).

Study participants and phases. We recruited 12 participants (3 female and 9 male) for the study via Slack outreach at Company – X. The participants had different technical roles and expertise, such as Research Engineers (principal, senior, and junior), Research Scientists (senior and junior), and Full-stack developers. The study was conducted by a co-author of the paper. And none of the paper authors were study participants. The study consisted of three phases: introduction, quiz, and survey. During the introductory

[5]The series labels are manually created by us based on the available table metadata.

[6]https://www.data.gov

Table 2: Example quiz tasks and their corresponding purposes and actions.

Questions	Purpose and actions
Q1: List all primary colors covered in the result tables.	`browse` tables →`lookup` column →`generate` set
Q2: Which animal type appears as the highest frequency in the most number of states?	`browse` tables →`lookup` column →`generate` aggregate→`compare`
Q3: What is the third most frequent status tag across all tables?	`browse` tables →`lookup` column →`generate` aggregate→`compare`
Q4: What is the average price of rabbits across all tables?	`browse` tables →`lookup` column→`identify` subset →`generate` aggregate
Q5: What is the min gross cost of the records with 10mg strength from the state of Florida?	`browse` tables →`lookup` column →`identify` subset→`generate` aggregate →`compare`
Q6: What is the second least frequent presentation among results in the year 2014?	`browse` tables →`lookup` column →`identify` subset→`generate` aggregate →`compare`

Table 3: Results of User Study. # Answers indicate how many users provided answers to the question; Accuracy means the portion of correct answers.

Metric	Method	Q1	Q2	Q3	Q4	Q5	Q6
Max Submission Time (s)	Notebook	336	509	364	613	173	216
	LakeVisage	**85**	**117**	**88**	**266**	**68**	**145**
Min Submission Time (s)	Notebook	87	70	42	144	64	**82**
	LakeVisage	**32**	**20**	**14**	**123**	**36**	85
Avg Submission Time (s)	Notebook	209.1	194.7	191.5	319.5	119.1	163
	LakeVisage	**54**	**62.7**	**46.5**	**172.8**	**51.75**	**111.4**
# Answered	Notebook	12	12	12	12	5	4
	LakeVisage	12	12	12	12	**12**	**12**
Accuracy (%)	Notebook	**100**	100	91.7	25	41.7	25
	LakeVisage	91.7	100	**100**	**100**	**100**	**100**

phase, participants received a brief overview of the study objectives and its follow-up phases. The quiz phase required participants to solve several tasks related to exploring data discovery results within a data lake using both a computational notebook for writing Python programs and LakeVisage. However, we alternated the order of systems between consecutive participants. Before answering the quiz with a system, participants were provided with a tutorial to help familiarize themselves with the functionalities — for LakeVisage, participants were introduced to various features; and for computational notebooks, participants were provided with a suite of utility functions to help in data discovery. Following the tutorial, participants performed several warm-up tasks before proceeding to the actual quiz tasks. Upon the completion of the quiz phase, participants were provided with a survey via Google Form to gauge their impressions about both systems.

Evaluation. We evaluated the completion time and accuracy for all of the tasks. We analyzed the survey responses to quantify the usability of both systems. We also collected qualitative feedback during the survey to understand the benefits and limitations of both systems. In presenting participant responses in Section 6.3, we included excerpts from complete responses (indicated by . . .). Additionally, we corrected spelling and grammatical mistakes. Therefore, the quotes presented in this paper are essentially paraphrases.

6.2 Results and Analysis

We analyze the quantitative and qualitative data collected during the quiz and interview phases to address our research questions.

Task Submission Performance. In Table 3, we show the response times of participants for each task, using computation notebook and LakeVisage, respectively. For all the tasks, participants' submission times using LakeVisage were obviously lower than notebook on average. Moreover, all of the participants completed at least five tasks in less time using LakeVisage. In fact, there were only two instances (Q4 by $P9$ and Q6 by $P10$) where participants took more time to complete tasks using LakeVisage.

Task Accuracy. As shown in Table 3, for the easier tasks (Q1-Q3), participants exhibited similar accuracy. Notably, among the participants who used notebooks, seven and eight did not complete submissions for the more challenging tasks Q5 and Q6, respectively. These tasks required participants to perform a sequence of operations on the data lake. However, to effectively implement those operations, participants had to formulate their understanding of the tables, relevant columns, and suitable sub-selections and then manually compare different candidates. Therefore, the errors in completing the eventual task may stem from the cognitive load associated with reasoning over the result of these operations. Though all the participants attempted Q4 when using a notebook, only 25% of the participants provided correct responses due to task complexity.

6.3 Participant Feedback

Our analysis of the survey results revealed that 83.3% of participants preferred LakeVisage to computational notebooks. These participants found LakeVisage visualizations useful in synthesizing information across multiple tables. On a scale of 1 to 5 (1=not challenging at all, 5=very challenging), the participants rated LakeVisage to be easier to use ($\mu = 2, \sigma = 0.27$) compared to notebooks ($\mu = 3, \sigma = 0.81$). We now discuss the strengths and limitations of both systems based on the qualitative feedback from participants.

Getting started with analytics sessions. Majority of participants ($N = 8$) found LakeVisage useful in launching the analytics sessions, specifically when working with unfamiliar datasets or a larger data collection — "... LakeVisage *provides the quick overview of data which is helpful to decide which direction we should go deeper*" ($P4$). In contrast, participants ($N = 7$) exhibited negative impressions towards notebooks when launching a session for the first time due the cumbersome interactions via iterative programming — "*you have to write some code before getting insights, might also require more understanding of the schema and values in the tables to write valid functions . . . takes a bit of trial-and-error . . .*" ($P12$).

Familiarity and learning curve. Besides the challenges in getting started, participants noted additional factors as potential limitations of notebooks such as lack of recall (of appropriate Dataframe operations) and a steeper learning curve (with literate programming). For example, $P1$ commented — "*Notebooks require users to be very familiar with Dataframes, while most users might forget if they don't use (Dataframes) frequently*". In contrast, participants found visualizations presented in LakeVisage intuitive. Three participants mentioned that LakeVisage would be very helpful for non-technical users due to the low barrier to familiarizing with visualizations — "*the visualization didn't need any prior familiarity*" ($P11$).

Convenience and ease of use. Participants ($N = 7$) also found LakeVisage easier to use compared with notebooks. Participants obtained insights faster with LakeVisage as shown in Table 3. For example, $P12$ commented — "*. . . faster to get insights, visualization tends to be more informative and provides much more insights than the*

notebooks". In contrast, participants deemed notebook operations time-consuming — "*Using Notebook took more time because I needed coding*" ($P2$). Participant $P10$ commented — "*... hard to manipulate (notebooks) unless you know (Dataframe) operations well*".

Flexibility of exploration. Participants ($N = 4$) appreciated the higher flexibility afforded by notebooks, specifically when performing more in-depth analysis. For example, new observations may prompt an analyst to issue new queries over the data lake on the fly. However, visualizations provided in LakeVisage are pre-defined and thus limit users' degrees of freedom. For example, participant $P12$ mentioned that notebooks provided "*a lot more flexibility in terms of analysis that can be done and analysts who are familiar might find it easier to use than learning to interact with a visualization tool*". Participant $P4$ commented — "*A notebook-based system enables us to conduct detailed and finer-grained operations if needed*". The same participant found LakeVisage to be limiting — "LakeVisage *sometimes lacks the customizability, especially when users want to execute complicated analysis*". Participant $P12$ mentioned — "*visualizations can sometimes be limited and less flexible when data is pre-aggregated in the visualization in a way that is far from the desired aggregation*".

Scope and functionalities. In terms of exploration goals, participants preferred visualizations to locate extremities in data and performing comparisons — "*I prefer Visualization for the purpose like finding the max/min values*" ($P2$). Participants preferred notebooks for dynamically computing aggregations over attributes of interest. Participant $P3$ found the top-K visualizations to be overwhelming due to the visual discontinuity caused by scrolling up and down the ranked list and requested features such as performing natural language querying over the visualizations. Two other participants requested similar features — "*For* LakeVisage*, having a simple search might be helpful. For example: show me any graphs that are related to a data column*" ($P8$). Participants ($N = 2$) also commented on the uncertainty introduced by approximations of the top-K results and requested greater transparency when communicating results.

7 RELATED WORK

7.1 Data Discovery from Data Lakes

There is a long stream of research works about data discovery from data lakes in the data management community. The examples of data discovery tasks include finding related tables [46], schema complement [26], domain discovery [37] and column annotation [31]. Among them the problem of finding related tables attracts more attentions in the recent years. There are two sub-tasks in this application, namely joinable table discovery and table union search [46]. Joinable table discovery aims at finding tables that can be joined with the given table on a column. LSH Ensemble [59] and JOSIE [58] employed syntactic similarity functions to decide joinability. PEXESO [8] computed the fuzzy similarity between columns based on pre-trained word, and Deep Join [9] relied on fine-tuned BERT model to capture the semantic information of columns. MATE [11] focused on the problem of composited join key with multiple columns. And Juneau [57] studied the joinable discovery over semi-structured data. The bottleneck of Table Union Search is to find a proper way to decide the column unionability scores. Nargesian et al. [36] proposed the first definition and design space for this problem. The D^3L system [3] divided columns into different categories and computed unionability score accordingly. SANTOS [24] utilized the knowledge base to decide unionability and Starmie [14] learned a BERT based encoder for table columns. Gen-T [12] tried to reclaim the original tables from unioned ones.

There are also some studies targeting at improving the usability of data lake related tasks. Aurum [15] provided a declarative query language to express data discovery tasks; RONIN [38] constructed a GUI to support navigation of data lakes [35] to ease the browsing. Ver [17] came up with a new variants of joinable table discovery to provide user-friendly views of data lakes. Auctus [6] aimed at improving the usability of keyword search over data lakes while Humboldt [2] automatically provided customized UIs for data discovery based on meta-data. None of above efforts can support our task that generates visualizations for results of data discovery tasks.

7.2 Visualization Recommendation Systems

Visualization recommendation systems can be broadly categorized into rule-based and machine learning (ML)-based approaches. Rule-based systems, such as Voyager [52], SeeDB [49], and AVA [50], rely on heuristics derived from expert experience or empirical studies. Technically, they shared the similar idea with earlier work of exploring results of OLAP queries [20, 44, 45]. These systems recommend visualizations based on specific goals or aesthetic properties. For example, Voyager suggests visualizations emphasizing visual appeal, while SeeDB highlights differences between datasets. LUX [28] enables visualization recommendations for dataframes in computational notebooks while factoring in user intent. Zenvisage [47] generalizes these systems by enabling the detection of desired visual patterns across large datasets. Additionally, systems like Profiler [22] and Scorpion [53] focus on identifying specific patterns, such as outliers, while VizDeck [23] presents a dashboard of potential 2D visualizations for a dataset. Transactional Panorama [48] studies the problem of refreshing visualization results. ML-based systems propose a different approach by learning from data instead of applying predefined rules. Draco-Learn [33] optimizes visualization recommendations by learning weights for trade-offs between design constraints. VizML [19] focuses on predicting design choices for visualizations, offering better interpretability and ease of integration. All these methods cannot operates over a set of result tables from data lakes w.r.t. the query table as LakeVisage did.

8 CONCLUSION

In this paper, we studied the new research problem of visualization recommendation for data discovery over data lakes. We first came up with a formal definition of this problem by addressing the issues of (i) defining how to build visualization over multiple result tables in the output of data discovery; and (ii) developing the concept of series to illustrate the relatedness in data discovery result; and (iii) handling multiple data formats including categorical, numerical and textual columns. Then, we proposed an end-to-end framework LakeVisage as the solution to this problem, which features the technical contributions of a data-driven approach to create the potential series of the visualization as well as progressive pruning strategies to remove unpromising visualization plans. Experimental results on end-to-end evaluation and user study demonstrated the efficiency and usability of our proposed framework.

REFERENCES

[1] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*. ACM, 29–42.

[2] Alex Bäuerle, Çagatay Demiralp, and Michael Stonebraker. 2024. Humboldt: Metadata-Driven Extensible Data Discovery. *CoRR* abs/2408.05439 (2024).

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *ICDE*. 709–720.

[4] Matthew Brehmer and Tamara Munzner. 2013. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Trans. Vis. Comput. Graph.* 19, 12 (2013), 2376–2385.

[5] Dan Brickley, Matthew Burgess, and Natasha F. Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *WWW*. 1365–1375.

[6] Sonia Castelo, Rémi Rampin, Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proc. VLDB Endow.* 14, 12 (2021), 2791–2794.

[7] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. *Proc. VLDB Endow.* 17, 8 (2024), 1925–1938.

[8] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *ICDE*. 456–467.

[9] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *Proc. VLDB Endow.* 16, 10 (2023), 2458–2470.

[10] Niklas Elmqvist and Jean-Daniel Fekete. 2010. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Trans. Vis. Comput. Graph.* 16, 3 (2010), 439–454.

[11] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *Proc. VLDB Endow.* 15, 8 (2022), 1684–1696.

[12] Grace Fan, Roee Shraga, and Renée J. Miller. 2024. Gen-T: Table Reclamation in Data Lakes. In *ICDE*. 3532–3545.

[13] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023. Table Discovery in Data Lakes: State-of-the-art and Future Directions. In *Companion of SIGMOD/PODS*. 69–75.

[14] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739.

[15] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *ICDE*. 1001–1012.

[16] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. Metam: Goal-Oriented Data Discovery. In *ICDE*. IEEE, 2780–2793.

[17] Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. 2023. Ver: View Discovery in the Wild. In *ICDE*. IEEE, 503–516.

[18] SC Gupta and VK Kapoor. 2020. *Fundamentals of mathematical statistics*. Sultan Chand & Sons.

[19] Kevin Zeng Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *CHI*. ACM, 128.

[20] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. 2015. Smart Drill-Down: A New Data Exploration Operator. *Proc. VLDB Endow.* 8, 12 (2015), 1928–1931.

[21] Minsuk Kahng, Shamkant B. Navathe, John T. Stasko, and Duen Horng (Polo) Chau. 2016. Interactive Browsing and Navigation in Relational Databases. *Proc. VLDB Endow.* 9, 12 (2016), 1017–1028.

[22] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Profiler: integrated statistical analysis and visualization for data quality assessment. In *AVI*. ACM, 547–554.

[23] Alicia Key, Bill Howe, Daniel Perry, and Cecilia R. Aragon. 2012. VizDeck: self-organizing dashboards for visual analytics. In *SIGMOD*. ACM, 681–684.

[24] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), 9:1–9:25.

[25] Albert Kim, Eric Blais, Aditya G. Parameswaran, Piotr Indyk, Samuel Madden, and Ronitt Rubinfeld. 2015. Rapid Sampling for Visualizations with Ordering Guarantees. *Proc. VLDB Endow.* 8, 5 (2015), 521–532.

[26] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *ICDE*. 468–479.

[27] Doris Jung Lin Lee, Himel Dev, Huizi Hu, Hazem Elmeleegy, and Aditya G. Parameswaran. 2019. Avoiding drill-down fallacies with *VisPilot*: assisted exploration of data subsets. In *IUI*. ACM, 186–196.

[28] Doris Jung Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2021. Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows. *Proc. VLDB Endow.* 15, 3 (2021), 727–738.

[29] Oliver Lehmberg and Christian Bizer. 2017. Stitching Web Tables for Improving Matching Quality. *Proc. VLDB Endow.* 10, 11 (2017), 1502–1513.

[30] Stephen Macke, Yiming Zhang, Silu Huang, and Aditya G. Parameswaran. 2018. Adaptive Sampling for Rapidly Matching Histograms. *Proc. VLDB Endow.* 11, 10 (2018), 1262–1275.

[31] Zhengjie Miao and Jin Wang. 2023. Watchog: A Light-weight Contrastive Learning based Framework for Column Annotation. *Proc. ACM Manag. Data* 1, 4 (2023), 272:1–272:24.

[32] Renée J. Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q. Pu, and Periklis Andritsos. 2018. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.* 41, 2 (2018), 59–70.

[33] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Trans. Vis. Comput. Graph.* 25, 1 (2019), 438–448.

[34] Kristi Morton, Magdalena Balazinska, Dan Grossman, and Jock D. Mackinlay. 2014. Support the Data Enthusiast: Challenges for Next-Generation Data-Analysis Systems. *Proc. VLDB Endow.* 7, 6 (2014), 453–456.

[35] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *SIGMOD*. ACM, 1939–1950.

[36] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825.

[37] Masayo Ota, Heiko Mueller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. *Proc. VLDB Endow.* 13, 7 (2020), 953–965.

[38] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proc. VLDB Endow.* 14, 12 (2021), 2863–2866.

[39] Karl Pearson. 1905. "Das fehlergesetz und seine verallgemeiner-ungen durch fechner und pearson." a rejoinder. *Biometrika* 4, 1-2 (1905), 169–212.

[40] Sajjadur Rahman, Maryam Aliakbarpour, Hidy Kong, Eric Blais, Karrie Karahalios, Aditya G. Parameswaran, and Ronitt Rubinfeld. 2017. I've Seen "Enough": Incrementally Improving Visualizations to Support Rapid Decision Making. *Proc. VLDB Endow.* 10, 11 (2017), 1262–1273.

[41] Sajjadur Rahman, Mangesh Bendre, Yuyang Liu, Shichu Zhu, Zhaoyuan Su, Karrie Karahalios, and Aditya G. Parameswaran. 2021. NOAH: Interactive Spreadsheet Exploration with Dynamic Hierarchical Overviews. *Proc. VLDB Endow.* 14, 6 (2021), 970–983.

[42] Sajjadur Rahman, Peter Griggs, and Çagatay Demiralp. 2021. Leam: An Interactive System for In-situ Visual Text Analysis. In *CIDR*. www.cidrdb.org.

[43] Aécio S. R. Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A Sketch-based Index for Correlated Dataset Search. In *ICDE*. 2928–2941.

[44] Sunita Sarawagi. 2000. User-Adaptive Exploration of Multidimensional Data. In *VLDB*. Morgan Kaufmann, 307–316.

[45] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. 1998. Discovery-Driven Exploration of OLAP Data Cubes. In *EDBT*, Vol. 1377. Springer, 168–182.

[46] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD*. 817–828.

[47] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya G. Parameswaran. 2016. Effortless Data Exploration with zenvisage: An Expressive and Interactive Visual Analytics System. *Proc. VLDB Endow.* 10, 4 (2016), 457–468.

[48] Dixin Tang, Alan D. Fekete, Indranil Gupta, and Aditya G. Parameswaran. 2023. Transactional Panorama: A Conceptual Framework for User Perception in Analytical Visual Interfaces. *Proc. VLDB Endow.* 16, 6 (2023), 1494–1506.

[49] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. 2015. SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *Proc. VLDB Endow.* 8, 13 (2015), 2182–2193.

[50] Jiazhe Wang, Xi Li, Chenlu Li, Di Peng, Arran Zeyu Wang, Yuhui Gu, Xingui Lai, Haifeng Zhang, Xinyue Xu, Xiaoqing Dong, Zhifeng Lin, Jiehui Zhou, Xingyu Liu, and Wei Chen. 2024. AVA: An automated and AI-driven intelligent visual analytics framework. *Vis. Informatics* 8, 1 (2024), 106–114.

[51] L Wasserman. 2013. All of statistics: A concise course in statistical inference.

[52] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock D. Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Trans. Vis. Comput. Graph.* 22, 1 (2016), 649–658.

[53] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proc. VLDB Endow.* 6, 8 (2013), 553–564.

[54] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*. ACM, 97–108.
[55] Beth Yost and Chris North. 2006. The Perceptual Scalability of Visualization. *IEEE Trans. Vis. Comput. Graph.* 12, 5 (2006), 837–844.
[56] Xiong Zhang, Jonathan Engel, Sara Evensen, Yuliang Li, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Teddy: A System for Interactive Review Analysis. In *CHI*. ACM, 1–13.
[57] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*. 1951–1966.
[58] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*. 847–864.
[59] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196.