LensXPlain: Visualizing and Explaining Contributing Subsets for Aggregate Query Answers

Zhengjie Miao, Andrew Lee, Sudeepa Roy
Duke University, Durham, NC
{zjmiao, andrew, sudeepa}@cs.duke.edu

ABSTRACT

In this demonstration, we will present `LensXPlain`, an interactive system to help users understand answers of aggregate queries by providing meaningful explanations. Given a SQL group-by query and a question from a user "*why output o is high /low*", or "*why output o_1 is higher/lower than o_2*", `LensXPlain` helps users explore the results and find subsets of tuples captured by predicates that contributed the most toward such observations. The contributions are measured either by *intervention* (if the contributing tuples are removed, the values or the ratios in the user question change in the opposite direction), or by *aggravation* (if the query is restricted to the contributing tuples, the observations change more in the same direction). `LensXPlain` uses ensemble learning for recommending useful attributes in explanations, and employs a suite of optimizations to enable explanation generation and refinement at an interactive speed. In the demonstration, the audience can run aggregation queries over real world datasets, browse the answers using a graphical user interface, ask questions on unexpected/interesting query results with simple visualizations, and explore and refine explanations returned by `LensXPlain`.

PVLDB Reference Format:
Zhengjie Miao, Andrew Lee, and Sudeepa Roy. LensXPlain: Visualizing and Explaining Contributing Subsets for Aggregate Query Answers. *PVLDB*, 12(12): 1898-1901, 2019.
DOI: https://doi.org/10.14778/3352063.3352094

1. INTRODUCTION

In today's world driven by data, many users with a variety of backgrounds seek to extract high level information from datasets by running aggregate queries. One common use case while exploring aggregate query answers is by asking '*why high/low*' questions on the outputs: e.g., '*why is the expenditure of Q1 of a retail chain high?*', or '*why is the profit of Q1 less than Q2?*'. While sophisticated visualization tools like Tableau [1] exist for exploring aggregates on a table, and OLAP operations like slice and dice or multi-dimensional aggregates by data cube have been studied in the database community, not much support exists for automatically finding explanations in response to such questions that are meaningful to a broad range of users.

To address this question, a formal framework of explanations was presented in [4] based on *intervention*: find summarized descriptions on input tuples by predicates as explanations such that if we intervene on the database and delete tuples depending on an explanation, the values of the data points mentioned in the user question change (e.g., if the question was '*why is the profit of Q1 less than the profit of Q2*', now the ratio of the profits of Q1 and Q2 is higher). The motivation comes from the literature on *causality*, a topic well-sudied in AI, Statistics, and Philosophy (see, *e.g.*, [3]), where by changing the value of a *cause*, the *effect* changes. Building upon this concept, in this demonstration we present `LensXPlain`, an interactive tool to automatically explain user questions by finding the subsets of tuples that contribute the most to the results mentioned in the question. The main features of `LensXPlain` are as follows:

1. **A simple visualization for aggregate group-by query answers and asking questions:** Before the user asks a question, the user needs to easily see the results and compare them. `LensXPlain` provides a simple interface to enter a SQL GROUP-BY query and visualize the results as a barchart as chosen by the user. Then the user can ask the 'why high/low' or 'why higher/lower' question by clicking on the bars or selecting from a list.
2. **Intuitive explanations by intervention and aggravation:** The explanations are predicates capturing a subset of tuples. The contribution of an explanation predicate is measured either by *intervention* (if the contributing tuples are removed, the values or the ratios in the user question change significantly in the opposite direction), or by *aggravation* (if the query is restricted to the contributing tuples, the observations change more in the same direction). `LensXPlain` returns top-k explanations by both these approaches.
3. **Visualizing the effect of an explanation:** `LensXPlain` '*explains an explanation*' by showing the effect of a selected explanation. If the user clicks on an explanation, she can see how the explanation changes the query answer by re-drawing the barchart.
4. **Refine explanations:** Since the explanations returned by an automated approach may not be 'absolutely correct', in `LensXPlain`, the user can give a

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352094

feedback using thumbs up/down for each explanation. The user feedback is used to refine the list (e.g., a thumbs down updates the list by removing similar explanations) as well as to update parameters in the explanation generation algorithm. The user can also select attributes she wants in the explanation, and add a limit on the size of the explanation predicates.

5. **Recommending attributes in explanations:** A real dataset can contain hundreds of attributes, which results in a huge search space for explanation predicates. The earlier work [4] used a small number of attributes hand-picked by the admin to generate the explanations. `LensXPlain` uses a novel attribute-recommendation approach to suggest attributes to the user that can generate explanations with high scores.

To support the above functionality at an interactive speed, `LensXPlain` uses a suite of algorithms and optimizations that we discuss in Section 2 along with the framework and implementation of `LensXPlain`. In Section 3 we present a walk-through of the proposed demonstration using the graphical user interface.

Related Work. In recent years, a number of visual data analytic tools such as Tableau, Voyager, and Zenvisage[1, 8, 6] have been introduced. SEEDB [7] enables users to explore SQL query results by recommending visualizations automatically. [9, 4] proposed frameworks for intervention-based explanations for outliers in aggregate query results. `LensXPlain` is inspired by [4], but it provides a first end-to-end easy-to-use system for automatically asking for, finding and exploring explanations. A detailed related work can be found in [4].

2. DETAILS OF LENSXPLAIN

Next we describe the framework of `LensXPlain`.

2.1 `LensXPlain` Framework

Given a relation R with attributes $A_1, ..., A_m$, `LensXPlain` takes as input an SQL aggregate query Q of the form[1]

```
SELECT Ai1, Ai2, ..., Aig, agg(a)
FROM R
GROUP BY Ai1, Ai2, ..., Aig
```

Here $A_{i_1}, A_{i_2}, ..., A_{i_g}$ are categorical attributes[2] and $dir \in \{high, low\}$. A user question indicates whether the user thinks the aggregate value of $t_1[agg(a)]_{Q,R}$ is higher/lower than expected, or respectively, $t_1[agg(a)]_{Q,R}$ is higher/lower than $t_2[agg(a)]_{Q,R}$. A **candidate explanation** ϕ is a conjunction of predicates on attributes: $\phi = \wedge_j \phi_j$, where each ϕ_j is an atomic predicate of the form $\phi_j = [A_i = c]$, such that A_i is an attribute of R and c is a constant. The *intervention* Δ_ϕ of ϕ denotes the set of tuples that satisfy ϕ, i.e., for all $t \in R \setminus \Delta_\phi$, $\phi(t) = false$. For better interpretability and practical performance, we introduce a parameter p to restrict that ϕ involves at most p atomic predicates.

Given the query Q and relation R, once the user asks a user question Γ, `LensXPlain` returns top-k explanations according to a scoring function μ for a user-specified parameter k. The scoring function $\mu(\phi)$ checks how an explanation ϕ affects the values of the tuple(s) specified in the user question Γ. The explanations can be of two types: (i) In the default **explanations by intervention**, the predicate ϕ characterizes a set of tuples whose *removal* will cause the aggregate value in the user question to change in the *opposite direction* of dir. For instance, if the user question was of the form why the value of t_1 is *high*, $\mu_{interv}(\phi) = -t_1[agg(a)]_{Q,R\setminus\Delta^\phi}$, i.e., if the intervention Δ_ϕ of ϕ significantly lowers the value of t_1, then the explanation ϕ has a high score. If the user question Γ involves two tuples t_1, t_2, and the question asks why the value of t_1 is higher than that of t_2, then $\mu_{interv}(\phi) = -\frac{t_1[agg(a)]_{Q,R\setminus\Delta^\phi}}{t_2[agg(a)]_{Q,R\setminus\Delta^\phi}}$, i.e., the explanations that significantly lower the ratio of t_1, t_2 are ranked higher. If $dir = low$ in Γ, the sign of the function μ is reversed. Intuitively, if the tuples satisfying the explanations were not there, that would significantly change the high or low values or ratios as observed by the user, thereby potentially contributing to the observation. Note that if the question involves only one tuple t_1, for a monotone query Q, we cannot get any meaningful explanations by intervention as by removing tuples the value of $t_1[agg(a)]$ can only be lower. To address this, as well as to provide another view of explaining the question asked by the user, `LensXPlain` also supports explanations by aggravation. (ii) In **explanations by aggravation**, the user is able to see what happens if the query is restricted only to the intervention Δ_ϕ of ϕ, i.e., $\mu_{aggr}(\phi) = t_1[agg(a)]_{Q,\Delta^\phi}$ or $= \frac{t_1[agg(a)]_{Q,\Delta^\phi}}{t_2[agg(a)]_{Q,\Delta^\phi}}$ if $dir = high$; the sign is reversed if $dir = low$. Intuitively, explanations by aggravation finds subset of tuples such that if the answer is restricted to that subset, the higher or lower value observed by the user is further aggravated, thereby potentially contributing to the observation of the user.

2.2 Algorithms and Optimizations

Challenges. One of the key goals of `LensXPlain` is to return top-k explanations (by intervention or aggravation) at an interactive speed. While the problem can be easily solved by a naive algorithm repeating over all possible explanations ϕ and reevaluating the query Q by removing or restricting to intervention Δ_ϕ, this approach is not sufficient due to the following challenges. **First,** Even for a relatively low value of maximum number of components p in the predicate ϕ, the number of potential predicates ϕ can be large, and running a for loop to go over the all candidate explanations takes a long time. **Second,** many datasets have a large number of attributes (e.g., the natality dataset used in [4] has more than 200 attributes). On the other hand, to ensure that the explanations are meaningful and to avoid overfitting, we want to limit the number of components p of explanations. `LensXPlain` helps users choose the most 'useful' attributes (or attributes according to the user's choice) for explanations out of many possible attributes.

Optimization with data cube. To circumvent the first challenge, [4] used the OLAP **data cube** operator supported by many standard database systems. The original aggregate query is translated into another aggregate query with cube, which incorporates all attributes that can participate in the explanation predicates in the `Group By` clause. As an example, suppose the user's aggregate query is:

```
Q1: SELECT A, B, count(*) as ct FROM R GROUP BY A, B
```

[1]The framework easily extends to filtering by WHERE clause and joins of multiple tables.

[2]Numerical attributes can be discretized by preprocessing.

Suppose the user selected a bar that corresponds to output tuple t_1 where $A = a$ and $B = b$, and asked why the original value of $t_1[ct] = x$ for this tuple is high. Let $E1, E2, E3$ be the attributes chosen for output explanations. Then the new aggregate query to generate the top-10 explanations is

```
Q2: SELECT TOP 10 E1, E2, E3, x - count(*) as newct
FROM R WHERE A = a and B = b
GROUP BY CUBE(E1, E2, E3)
ORDER BY newct ASC
```

The rows in the output of query $Q2$ will have the new value of query $Q1$ when tuples satisfying the corresponding predicate of the form $[E1 = e1$ and $E2 = e2$ and $E3 = e3]$ are removed from the database; the data cube operator allows the predicates to ignore some attributes, *e.g.*, predicates of the form $[E1 = e1$ and $E3 = e3]$ may also be returned. Since the user asked why the original value of the query $Q1$ at $A = a, B = b$ (*i.e.*, x) was high, the `ORDER BY` clause in query $Q2$ favors explanation predicates that lower the value of x to a large extent and thereby explain this observation.

Attribute selection. The use of data cubes enables `LensXPlain` to efficiently evaluate the scores of a huge number of potential explanation predicates in real time. However, further optimizations are still needed to address the second challenge of useful attribute selection. The framework proposed in [4] does not address this question as the attributes were hand-picked before running the explanation generation algorithm. `LensXPlain` employs a combination of two approaches to find relevant attributes for explanations.

(1) Using association rule mining. The first one is inspired by the Association Rule Mining problem. Association rule mining searches for interesting relationships between attributes in a database. A rule $X \Rightarrow Y$ indicates that the itemset X is related to the itemset Y. By regarding values of attributes as items, we apply the classic Apriori algorithm [2] in `LensXPlain` to find relevant attribute values to the groups in the user question.

(2) Using a random forest classifier. While the first approach models the attribute recommendation as an unsupervised learning problem, our second approach models it as a supervised learning problem. Consider the user question involving two values in the query result that asks why the count corresponding to $A = a1$ and $B = b1$ is high compared to the count corresponding to $A = a2$ and $B = b2$. There are two groups in the question, and we can regard each group as a class and each tuple belonging to these two groups in the original relation as an observation for the class. Therefore, the attribute recommendation problem is transformed to finding the attributes that are most important for discriminating these two classes. In `LensXPlain`, we used Random Forests of Scikit Learn and used its built-in measure for variable importance to find relevant attributes.

An ensemble learner for attribute selection. For user questions involving comparison of two result tuples, we ensemble the rule mining approach and the random forest approach in a boosting manner (scores from the Random Forest approach are normalized; since we do not have testing data here, initially, we just assign weights between them equally). Once the explanations are shown, we adjust the weights from the user's feedback on the explanations. For user questions including only one result tuple, the recommendation is based on the result of association rule mining.

Other optimizations including discarding candidate predicates with very low support values and caching a result set in the frontend are also deployed.

Implementation of `LensXPlain`. `LensXPlain` uses PostgreSQL 10.4 as the DBMS and the backend server is implemented in Python 3.6 and Flask; the GUI is implemented using Javascript and Vega-Lite [5].

3. DEMONSTRATION

The goals of our demonstration are (1) to illustrate that the user interface of `LensXPlain` provides an interactive mechanism to explore aggregate query results, and more importantly, (2) to explore explanations interactively with the help of the optimizations proposed in Section 2.

Datasets. We will illustrate `LensXPlain` using multiple public real datasets: *Natality dataset* (birth records in the USA in a year, ~4M rows and ~200 attributes), the NSF grant dataset (~1.4M rows in 3 tables and ~25 attributes), and the Adult dataset (~48K rows and ~14 attributes)[3].

User interface. The web UI of `LensXPlain` shown in Figure 1 has five main components: $\boxed{a}$**:** list of the available tables, schema of current table, and buttons for attribute recommendation and table content view, $\boxed{b}$**:** SQL query constructor, $\boxed{c}$**:** visualization for query answers, $\boxed{d}$**:** user question constructor and status panel, and $\boxed{e}$**:** list of explanations. The user can interact with `LensXPlain` as follows:

(i) Forming a SQL Group-by query and visualizing the answers. The list of the available tables (and for a table, the list of attributes) will be shown on the left pane $\boxed{a}$, where the user can also view the content of the table. In $\boxed{b}$, using a pre-populated query template, the user can choose a table, type in the group-by attributes, and add a where clause by clicking on the "+" button. After submitting the query, a grouped barchart will be displayed in the center view $\boxed{c}$. By default, each value of the first group-by attribute corresponds to a group of bars, and each value of the second group-by attribute corresponds to an individual bar (x-axis) as well as the color, and the the height of the bars (y-axis) represents aggregate values. Furthermore, query results with more than two group-by attributes are also supported. The user can click on the "Visualization Settings" button to update the column/x-axis/color attributes in the visualization in the pop-up window. If there are more than 3 group-by attributes, the user can also select values of rest attributes in the pop-up. The color attribute can be different from the column attribute and x-axis attribute, and then a stacked barchart will be displayed.

(ii) Forming User Question. After the barcharts are displayed, the user is able to explore the result and seek explanations on unexpected/interesting results. For instance, in Figure 1, the group-by query on `income_group, race` plots the number of people in high-income group and low-income group by different races. Suppose the user observes that the number of high-income black people is low compared to the number of high-income white people. To find most significant sub-populations contributing to this observation, she can simply click on these bars as shown in Figure 1 (the bars are highlighted upon clicking, and the value of the drop-down menu in the user question constructor $\boxed{d}$, changes accordingly). Then she selects 'high' as the direction, and enters top-10 for parameter k. A parameter for the

[3] `http://www.cdc.gov/nchs/data_access/ftp_data.htm`, `https://www.nsf.gov/awardsearch/download.jsp`, `https://archive.ics.uci.edu/ml/datasets/adult`

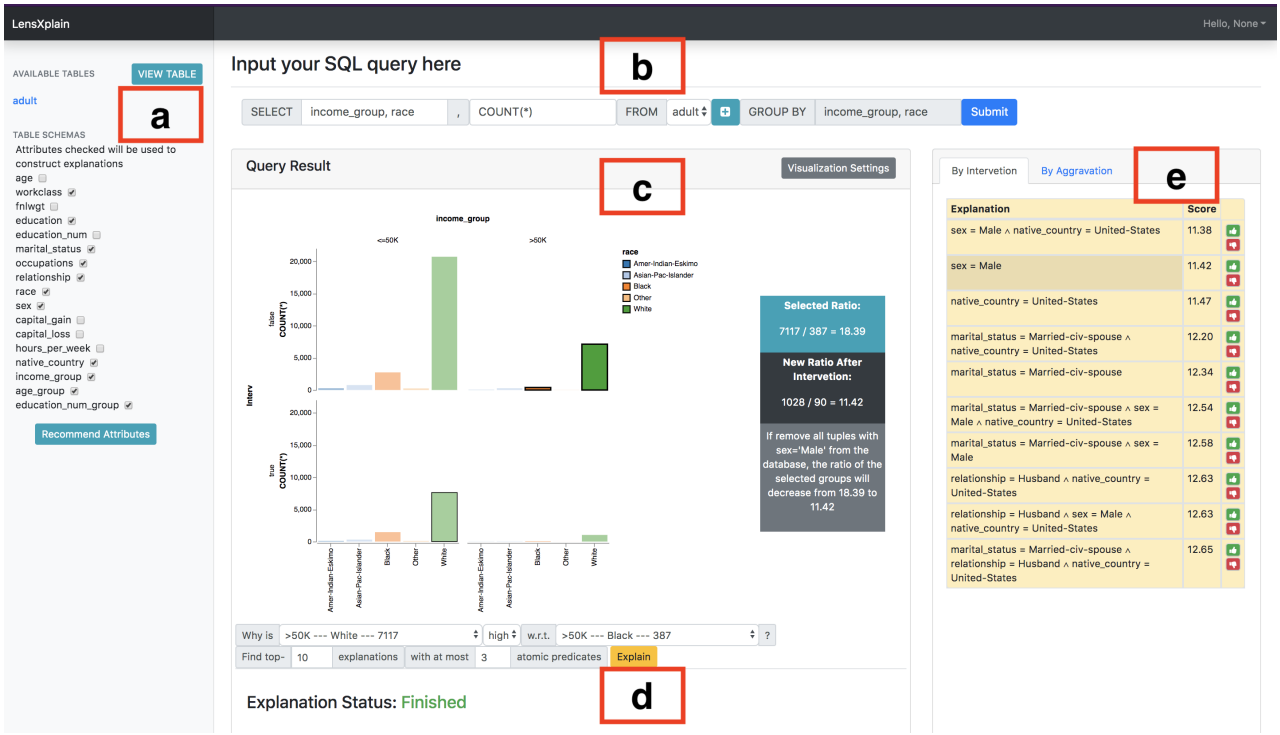

Figure 1: **Web user interface of `LensXPlain`.**

number of atomic predicates can also be set. Although in this example, the two groups correspond to the same value of attribute 'income_group', in general, the selected bars in the two groups can be arbitrary, and the user is allowed to ask questions on a single group or bar as well.

(iii) Explanations and 'explaining explanations'. Once the user question is entered and the 'Explain' button is clicked, the top-k explanations returned by the algorithm are displayed on the right side $\boxed{e}$ in decreasing order of their *scores*. The user is allowed to choose either explanation *by intervention* or *by aggravation* from two tabs, and then click on one explanation to dive into. A new barchart of the intervened or aggravated query result will be plotted below the original barchart, and the new value/ratio of the selected query answers will also be displayed, along with a natural language statement that describes how removing or restricting to the set of tuples represented by the selected explanation modifies the query result (see Section 2). This facilitates comparison between the query results before and after the intervention or aggravation for the selected explanation predicate is enforced.

(iv) Refining explanations. There are two thumbs up/down buttons next to each explanation. If the user thumbs down an explanation, then all explanations containing this predicate will be removed from the list of explanations, and the list will be updated. The user can keep crossing out explanations she does not consider useful, until a new user question is submitted.

(v) Recommending relevant attributes. As we discussed in Section 2, we want to restrict the number of attributes used in the explanations to improve the performance of explanation generation. Before the user clicks the "Explain" button, she can choose which attributes to be considered in the explanations by selecting the checkboxes in the left-hand-side view $\boxed{a}$. Moreover, she can click "Recommend attributes" to see what attributes are recommended by our algorithm, and then decide whether to include them.

Acknowledgements. This work is supported in part by NSF Awards IIS-1552538, IIS-1703431, and NIH award 1R01EB025021-01.

4. REFERENCES

[1] `http://www.tableausoftware.com/`.
[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.
[3] J. Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, 2000.
[4] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. *SIGMOD*, pages 1579–1590, 2014.
[5] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2017.
[6] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.
[7] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.
[8] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658, 2016.
[9] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.