# Selecting Sub-tables for Data Exploration

Yael Amsterdamer
Bar-Ilan University

Susan B. Davidson
University of Pennsylvania

Tova Milo
Tel Aviv University

Kathy Razmadze
Tel Aviv University

Amit Somech
Bar-Ilan University

**Abstract**

Data scientists frequently examine the raw content of large tables when exploring an unknown dataset. In such cases, small subsets of the full tables (*sub-tables*) that accurately capture table contents are useful. We present a framework which, given a large data table $T$, creates a sub-table of small, fixed dimensions by selecting a subset of $T$'s rows and projecting them over a subset of $T$'s columns. The question is: Which rows and columns should be selected to yield an *informative* sub-table?

Our first contribution is an informativeness metric for sub-tables with two complementary dimensions: *cell coverage*, which measures how well the sub-table captures prominent data patterns in $T$, and *diversity*. We use association rules as the patterns captured by sub-tables, and show that computing optimal sub-tables directly using this metric is infeasible. We then develop an efficient algorithm that indirectly accounts for association rules using *table embedding*. The resulting framework produces sub-tables for the full table as well as for the results of queries over the table, enabling the user to quickly understand results and determine subsequent queries. Experimental results show that high-quality sub-tables can be efficiently computed, and verify the soundness of our metrics as well as the usefulness of selected sub-tables through user studies.

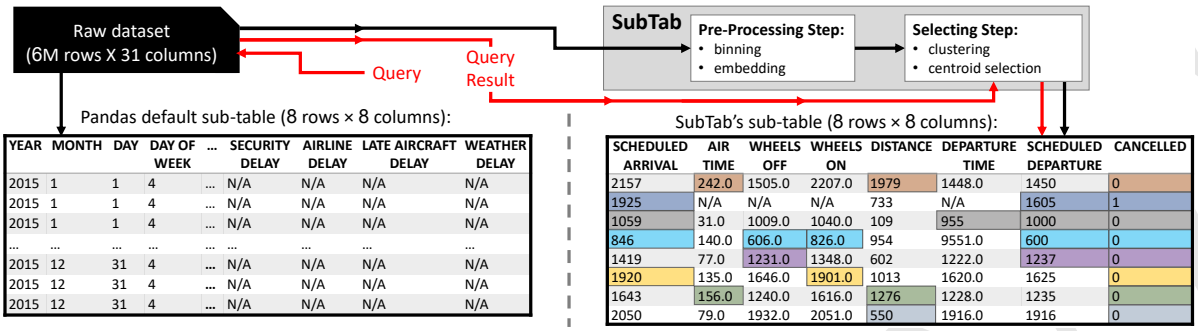
## 1 Introduction

Data exploration is the process of understanding an unfamiliar dataset and determining what part of the data is relevant to an analysis task. The process is often done by iteratively applying exploratory queries, examining their results, and interpreting them with data visualizations.

Numerous lines of work have studied solutions for different aspects of the exploration process, such as query suggestions (e.g., [46, 52]), query formulation for non-programmers [16, 19, 53, 57], insights-discovery [35, 36, 58], and visualization recommendations [39, 42, 63].

In this work, we focus on the task of *tabular data display*: before applying additional queries or visualizations, users often manually examine the raw table or query results. This is done using commands such as the table display of the popular Pandas library[1], which shows a small subset of the raw table rows and columns that is easier for a human user to inspect than the full table. However, a disadvantage of the Pandas tabular display is that its choice of rows and columns to display is arbitrary: it includes

[1] Pandas: Python Data Analysis Library. `https://Pandas.pydata.org/`




| YEAR | MONTH | DAY | DAY OF WEEK | ... | SECURITY DELAY | AIRLINE DELAY | LATE AIRCRAFT DELAY | WEATHER DELAY |
|---|---|---|---|---|---|---|---|---|
| 2015 | 1 | 1 | 4 | ... | N/A | N/A | N/A | N/A |
| 2015 | 1 | 1 | 4 | ... | N/A | N/A | N/A | N/A |
| 2015 | 1 | 1 | 4 | ... | N/A | N/A | N/A | N/A |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2015 | 12 | 31 | 4 | ... | N/A | N/A | N/A | N/A |
| 2015 | 12 | 31 | 4 | ... | N/A | N/A | N/A | N/A |
| 2015 | 12 | 31 | 4 | ... | N/A | N/A | N/A | N/A |

| SCHEDULED ARRIVAL | AIR TIME | WHEELS OFF | WHEELS ON | DISTANCE | DEPARTURE TIME | SCHEDULED DEPARTURE | CANCELLED |
|---|---|---|---|---|---|---|---|
| 2157 | 242.0 | 1505.0 | 2207.0 | 1979 | 1448.0 | 1450 | 0 |
| 1925 | N/A | N/A | N/A | 733 | N/A | 1605 | 1 |
| 1059 | 31.0 | 1009.0 | 1040.0 | 109 | 955 | 1000 | 0 |
| 846 | 140.0 | 606.0 | 826.0 | 954 | 9551.0 | 600 | 0 |
| 1419 | 77.0 | 1231.0 | 1348.0 | 602 | 1222.0 | 1237 | 0 |
| 1920 | 135.0 | 1646.0 | 1901.0 | 1013 | 1620.0 | 1625 | 0 |
| 1643 | 156.0 | 1240.0 | 1616.0 | 1276 | 1228.0 | 1235 | 0 |
| 2050 | 79.0 | 1932.0 | 2051.0 | 550 | 1916.0 | 1916 | 0 |

Figure 1: System Workflow. SubTab takes as input a raw table used throughout the EDA session. The sub-table on the left is the output of the display command in Pandas, a random subset of the table's rows and columns. The sub-table on the right is the output of SubTab, in which some of the ARs are highlighted for purposes of illustration. SubTab has two-steps: (1) Pre-Processing Step - the table is binned and embedded into vectors; (2) Selecting Step - triggered by a display of the input table or as the result of a query over the table, an *informative sub-table* is displayed. This is done by clustering the row and column vectors of the query result in the embedded space and selecting the centroids as representatives.

a few of the first and last rows and columns. As a result, the sub-tables may include arbitrary data values and cross-column value combinations rather than ones that reflect prominent trends; it may also elide columns that are important for further exploration and analysis.

**Example 1.1.** *Consider a table $T$ taken from the Kaggle* flights *dataset [2] which contains 31 columns and ∼6M rows. $T$ may be used for analysis tasks such as predicting flight delays and cancellations. The analyst starts by inspecting the data using* Pandas display($T$)*, which yields the sub-table displayed at the top left of Figure 1. The usefulness of this display for data exploration is intuitively limited: for instance, the first rows, last rows, and last columns are repetitive, and the displayed values are arbitrary. In addition, the chosen columns may not be relevant for the analyst's goals: for instance, the analyst may be interested in flight cancellations, yet there is no reason to assume that the columns in the* Pandas *sub-table are relevant for this goal.* □

Our goal is to develop a foundational approach for an informed selection of sub-tables that characterizes a given table as a whole and captures multiple diverse patterns. Concretely, given a table $T$ with $n$ rows and $m$ columns, our goal is to create a sub-table $T_{\text{sub}}$ with $k << n$ rows and $l << m$ columns which is a subset of $k$ rows projected over a subset of $l$ columns of $T$. To do this, we introduce a novel notion of sub-table informativeness which reflects (1) the extent to which the sub-table captures prominent *data patterns* and trends in the full table, including cross-column patterns. This is based on *association rules* (ARs) as a standard type of pattern that applies to a subset of a row's cells and hence allows several such patterns to be captured is a small sub-table. (See a discussion of other types of patterns in Section 7.); (2) the *diversity* in the sub-table, including representative values from each selected column and minimizing data repetitions. We also allow users to specify *target columns* that affect the choice of sub-tables so that it captures columns and ARs relevant to the target column.

**Example 1.2.** *Continuing with the* flights *dataset, an* informative *sub-table is shown at the right of Figure 1. Unlike the sub-table at the top left, this sub-table captures several prominent ARs that hold over the input table and that include the target column*

*CANCELLED. For purposes of illustration, we show this by highlighting cells that participate in one of the ARs that hold for the row (many additional ARs hold). For example, the highlighted rule in the first row states that long flights (AIR_TIME*$\in [198.0, 422.0]$ *and DISTANCE*$\in [1546.0, 2724.0]$*) are likely* not *to be cancelled (CANCELLED*$= 0$*). The highlighted rule in the second row states that short afternoon flights (according to the SCHEDULED_DEPARTURE and SCHEDULED_ARRIVAL columns) are likely to be canceled. The sub-table also shows diverse values per column by including representatives of diverse sub-groups in the value distribution, e.g., morning, afternoon and night arrivals/departures, short, medium and long flight distances, etc.* □

We formalize (Section 2) our notion of informativeness based on a combination of two complementary metrics: *cell coverage* and *diversity*. Cell coverage measures how well the sub-table $T_{\text{sub}}$ represents data patterns in the full table $T$, which are defined via ARs. Concretely, given a set of prominent ARs that hold over the full table $T$,[2] our metric reflects the number of cells in $T$ that are describable by ARs that are captured in $T_{\text{sub}}$. If one or more *target columns* are known in advance to be the focus of the analysis, they will be included in the $l$ selected columns, and we measure cell coverage only according to ARs that include one or more target columns. For the second metric, *diversity*, we split the value distribution of a column (continuous or categorical) into meaningful *bins*. We then compute a Jaccard-like row similarity score based on these bins, and use the average pairwise row distance as the diversity score. Intuitively, in a small sub-table we cannot capture the full distribution of each column, and our metric is geared towards increasing the number of represented bins and reducing bin repetitions. We use a score combining these metrics to measure the informativeness of a sub-table.

Unfortunately, we show (Section 3) that optimizing our informativeness measures directly is generally infeasible, and discuss the limitations of approximate solutions. In particular, although there are several efficient techniques for mining ARs (e.g., [13, 32, 47, 55, 56]), they are still too time-consuming for large datasets in an interactive setting.

We therefore propose a sub-table selection method which *indirectly* accounts for ARs using *table embedding* [20, 21], and that allows us to avoid calculating ARs thereby speeding up and simplifying the entire process (Section 4). Concretely, given a table $T$ we use *binning* [55] to split each column's values into a small set of meaningful groups. We then compute an embedding of table cells as vectors. As we demonstrate in Section 4.2, the embedding captures bin co-occurrences, and therefore implicitly corresponds to ARs. To select rows and columns for a sub-table we derive from the cell vectors a vector representation for rows and columns, cluster them (separately), and select the centroids as rows and columns that represent diverse patterns in the data.

An important benefit of our solution design is in *responding to queries over* $T$: during the exploratory data analysis (EDA) session, users typically issue different queries on a given table $T$ (red arrows of Figure 1). Our computation of cell embedding may be viewed as a part of the pre-processing step of a given data table $T$, along with the binning of its values (first blue box in Figure 1). This step is executed once upon loading the table. Then, for subsequent selection-projection query $Q$ on $T$, we can compute the vector representation of rows and columns in $Q(T)$ based on the cells of $T$ that appear in them, and re-execute clustering and centroid selection (*Selecting* step in Figure 1, shown as the second blue box). This significantly speeds up sub-table computation compared with computing everything from scratch (a few seconds instead of up to a minute for large tables).

[2] There are standard metrics we can use to measure the prominence of ARs in $T$, such as Support and Confidence [13]. Also see Section 5.4.

**Contributions.** Main contributions of this work include:

1. A formal notion of *informativeness* for sub-tables, designed to measure how well a sub-table characterizes the full table. The capturing of prominent data patterns in the sub-table is measured by *cell coverage* (for ARs), while the ratio of non-repetitive, representative values is measured by *diversity*.

2. A *complexity analysis* of the problem of selecting an optimal sub-table, which shows the infeasibility of computing sub-tables that optimize our informativeness metrics.

3. A *greedy algorithm* (CCSG), which traverses column combinations, and greedily selects rows for each such combination. This algorithm approximates the optimal cell coverage if all column combinations can be traversed within a given time limit. We also present RCCSG, a variant that samples rows to speed up row selection. We discuss the practical limitations of both algorithms, and use them as baselines against which we compare the quality of computed sub-tables.

4. A *practical algorithm*, SubTab, for computing informative sub-tables, which accounts for ARs indirectly using table embedding. The algorithm has two phases: a pre-processing phase that performs binning and embedding, and can be executed as soon as the data is loaded; and a clustering and centroid selection phase that is called for each sub-table display, e.g., on query results.

5. An *implementation* for SubTab as a local Python library that hooks into Pandas, and displays tables and query results as informative sub-tables. Its UI includes optional highlighting of ARs (as shown in Figure 2).

6. *Experimental results* that measure the sub-table quality and running time of SubTab compared with several baselines. Among the interactive time algorithms, SubTab achieves the best quality by a large gap. Among the algorithms that achieve the best quality, including ones that directly optimize our metric, SubTab is the only one which executes in interactive speed. Experiments with real EDA sessions and user studies show that our quality metrics are compatible with both subjective user experience and objective performance in insight discovery.

**Organization.** The rest of the paper is organized as follows. We start in Section 2 by defining our metrics of cell coverage and diversity. Section 3 gives hardness results for the problem of optimal sub-table selection and greedy algorithms. Section 4 presents our sub-table computation method based on table embeddings. Section 5 describes our experimental study. Related work is discussed in Section 6. We conclude in Section 7.

# 2 Model and Metrics

In this section, we formalize the notion of "sub-table informativeness" via metrics of cell coverage and diversity and define an optimization problem based on their combination.

## 2.1 Model

A relational schema $U = \{u_1, \ldots, u_{|U|}\}$ is a finite set of columns, such that each column $u_i$ allows values from a subset of the global domain $\mathcal{D}_i \subseteq \mathcal{D}$ (e.g., for a binary column, $\mathcal{D}_i = \{0, 1\}$). A relational table over $U$ is a finite set $T \subseteq U \to \mathcal{D}$ of tuples such that $t(u_i) \in \mathcal{D}_i$ is the value of the cell in the row corresponding to a tuple $t \in T$ and the column $u_i \in U$.

```
from  SubTab import subTab
sdf = subTab(flights_df, use_rules=True)
df1 = flights_df[(flights_df['DISTANCE']>2000) & (flights_df['DEPARTURE_DELAY']>10)]
sdf.display(df1)
```

For creating new table vectors, it took 0:00:02.282
For summary creation, it took 0:00:00.45
{'cell_cov': 0.31. 'jaccard':0.72}

| Destination Airport | Arrival Time | Wheels On | Airline Delay | Departure Time | Wheels Off | Scheduled Departure | Late Aircraft Delay | Distance | Departure Delay |
|---|---|---|---|---|---|---|---|---|---|
| MCO | 1842.0 | 1832.0 | 102.0 | 1050.0 | 1107.0 | 855 | 0.0 | 2218 | 115.0 |
| JFK | 742.0 | 736.0 | N/A | 2333.0 | 2345.0 | 2303 | N/A | 2475 | 28.0 |
| Rule #2: {(Destination Airport -'SEA'), (Arrival Time -'high'), (Distance -'high')} →{(Airline Delay -'low'), (Departure Delay -'high')}, support-0.11, confidence-0.5, lift-1.5 | | | | | | | | | |
| SEA | 2300.0 | 2253.0 | 20.0 | 1939.0 | 2012.0 | 1807 | 72.0 | 2402 | 92.0 |
| SAN | 1941.0 | 1938.0 | N/A | 1649.0 | 1702.0 | 1629 | N/A | 2588 | 20.0 |
| BOS | 1550.0 | 1545.0 | N/A | 752.0 | 806.0 | 730 | N/A | 2611 | 22.0 |
| SFO | 2136.0 | 2130.0 | N/A | 1820.0 | 1834.0 | 1808 | N/A | 2704 | 12.0 |
| JFK | 16.0 | 3.0 | 0.0 | 1603.0 | 1615.0 | 1535 | 17.0 | 2475 | 28.0 |

Figure 2: Informative 10X10 sub-table for a large query result

**Definition 2.1** (Sub-table)**.** *Given a table $T$ over schema $U$, a sub-table $T_{sub}$ is a table over schema $U_{sub} \subseteq U$ such that each tuple $t \in T_{sub}$ is the projection of some tuple $t' \in T$ over the columns of $U_{sub}$, i.e., for every $u \in U_{sub}$, $t(u) = t'(u)$.*

We next define two standard notions that will be useful in the sequel: binning and ARs.

In a schema $U$, each column $u_i$ may be *categorical*, namely, $\mathcal{D}_i$ is discrete, e.g., a column of airline names; or *continuous*, namely, $\mathcal{D}_i$ is a continuous range, e.g., a column of flight distance. Moreover, in a table $T$ over $U$ a different distribution of values (e.g. uniform or skewed) may occur in each column. *Binning* the column values is a technique commonly used to allow a uniform treatment of columns with different ranges and distributions. Formally,

**Definition 2.2** (Binning)**.** *Given a table $T$ over schema $U$, a* binning *function $\mathcal{B}$ maps each column $u_i \in U$ to a finite set of bins $\mathcal{B}_i = \{B_1^i, \ldots, B_{|\mathcal{B}_i|}^i\}$ such that for every $t \in T$, $t(u_i)$ belongs to exactly one bin $B_j^i \in \mathcal{B}_i$.*

**Example 2.3.** *In the flights' dataset, we split the (continuous) range of the* DISTANCE *column into the bins short, medium, and long-distance. Depending on the column value distribution, we may obtain $B_{long}^{DIST} = [1546.0, 2724.0]$. The* CANC. *column is binary, hence we can use its categories as bins.*

Next, we recall the notion of *association rules* (ARs), which we use to capture patterns in the data. ARs will be used to measure and compare the quality of sub-tables. Formally,

**Definition 2.4** (Association rules (ARs) [55])**.** *Given a table $T$ over schema $U$, an AR $R$ has the form $\{(u_1, v_1), \ldots (u_r, v_r)\} \rightarrow \{(u_{r+1}, v_{r+1}), \ldots (u_{r+p}, v_{r+p})\}$ where each $u_i \in U$ is a column and each $v_i \in \mathcal{D}_i$ is a cell value. Denote by $U_R = \{u_1, \ldots, u_{r+p}\} \subseteq U$ the set of columns used in $R$. We say $R$* holds *for a tuple $t \in T$ if $t(u_i) = v_i$ for every $1 \leq i \leq r + p$. Denote by $T_R \subseteq T$ the subset of tuples for which $R$ holds.*

| $\hat{T}$ | ROW | CANC. | DEP._TIME | MONTH | SCHED._DEP. | DISTANCE |
|---|---|---|---|---|---|---|
| | 1 | 1 | N/A | 6-8 | afternoon | short |
| | 2 | 1 | N/A | 6-8 | afternoon | medium |
| | 3 | 1 | N/A | 6-8 | morning | medium |
| | 4 | 1 | N/A | 6-8 | morning | short |
| | 5 | 0 | morning | 9-11 | morning | medium |
| | 6 | 0 | morning | 6-8 | morning | medium |
| | 7 | 0 | evening | 6-8 | evening | long |
| | 8 | 0 | evening | 6-8 | afternoon | long |

| $\hat{T}^{(1)}_{\text{sub}}$ | ROW | CANC. | DEP._TIME | MONTH | DISTANCE |
|---|---|---|---|---|---|
| | 1 | 1 | N/A | 6-8 | short |
| | 5 | 0 | morning | 9-11 | medium |
| | 7 | 0 | evening | 6-8 | long |

| $\hat{T}^{(2)}_{\text{sub}}$ | ROW | CANC. | DEP._TIME | MONTH | SCHED._DEP. |
|---|---|---|---|---|---|
| | 1 | 1 | N/A | 6-8 | afternoon |
| | 5 | 0 | morning | 9-11 | morning |
| | 7 | 0 | evening | 6-8 | evening |

Figure 3: Example Table $\hat{T}$ with two sub-tables. One AR per row is highlighted by (arbitrary) colors and underlines

The use of binning may improve mined ARs [55]: given a table $T$, we can replace each cell value $t(u_i)$ with an identifier of its matching bin $B^i_j$. Consequently, one may be able to mine ARs that apply to more tuples.

**Example 2.5.** *Using the bins from example 2.3, the AR from Example 1.2 stating that long flights are likely* not *to be cancelled can be written as:*
$\textit{AIR\_TIME} \in B^{AT}_{long}, \textit{DISTANCE} \in B^{DIST}_{long} \nrightarrow \textit{CANC.} \in B^{CANC}_{0}$.

## 2.2 Informativeness Metrics

We now develop quality metrics for sub-tables. Out of various possible informativeness metrics, we base our choice on the role of a sub-table as a compact view of the raw data that characterizes the table as a whole. The first type of metric that we develop intuitively measures how well data patterns in the full table are captured by the sub-table. Specifically, we focus on ARs, which apply to a subset of a row's cells and hence several such patterns can be captured in a small sub-table.

**Cell coverage.** Given a sub-table $T_{\text{sub}}$ of table $T$ and a set $\mathcal{R}$ of ARs mined from $T$ (e.g., using [55]), we ask:

1. Which ARs of $\mathcal{R}$ are *covered*, i.e., captured by $T_{\text{sub}}$?
2. What is the marginal contribution of each covered AR to $T_{\text{sub}}$'s informativeness?
3. How do marginal contributions aggregate to an overall numerical score for $T_{\text{sub}}$?

Since sub-tables include a subset of the table cells, and ARs are also defined at the level of table cells, we propose below formal definitions for q1-q3 that yield a

*cell coverage* metric. This metric intuitively reflects the ratio of cells in $T$ that are describable by ARs in $\mathcal{R}$ that are represented in $T_{\text{sub}}$.

**Definition 2.6** (Cell coverage)**.** *Let $T$ be a table, $\mathcal{R}$ a set of ARs mined from $T$, and $T_{sub}$ a sub-table of $T$.*

1. *An AR $R \in \mathcal{R}$ is said to be* covered *by $T_{sub}$ if all the attributes of $R$ are in $T_{sub}$ ($U_R \subseteq U_{sub}$), and there exists a tuple $t \in T_{sub}$ for which $R$ holds ($\{T_{sub}\}_R \neq \emptyset$). Let $\mathcal{R}_{sub}$ be the subset of $\mathcal{R}$ that is covered by $T_{sub}$.*

2. *The* marginal contribution *of $R \in \mathcal{R}_{sub}$ is the subset of table cells it describes:* $\text{cell}(R, T) := \{\langle t, u\rangle \mid t \in T_R \wedge u \in U_R\}$.

3. *The* cell coverage *of $T_{sub}$ w.r.t. $T, \mathcal{R}$ is denoted by*

$$\text{cellCov}_{\mathcal{R}}(T, T_{sub}) := \frac{1}{\text{upcov}} \left| \bigcup_{R \in \mathcal{R}_{sub}} \text{cell}(R, T) \right| \tag{1}$$

   *I.e., it is the (normalized) number of cells in $T$ described by any covered rule in $R \in \mathcal{R}_{sub}$. The normalization factor* $\text{upcov} := \left|\bigcup_{R \in \mathcal{R}} \text{cell}(R, T)\right|$ *is an upper bound on the number of cells that can be covered, and ensures that* $\text{cellCov}_{\mathcal{R}}(T, T_{sub}) \in [0, 1]$.

We next motivate our choice of cell coverage metric by a brief overview of alternative approaches, through an example.

**Alternative coverage metrics.** Table $\hat{T}$, on the left of Figure 3, illustrates some of the trends in the *Flights* dataset mentioned above, with *CANC.* as the target column. The table values represent bin names (e.g. "short", "medium", "long"). ARs of maximal size are highlighted; each highlighted line illustrates a different rule, with colors alternating for clarity.

First, observe that rows with *CANC.*=1 are more homogeneous compared with rows with *CANC.*=0 due to the fact that many fields are not applicable when a flight is canceled. As a result, there are 13 ARs for the first 4 rows, and only 8 for the last 4. This issue is exacerbated for larger homogeneous data subsets, leading to many overlapping rules including subsets of their values. We therefore propose to use measures based on *data coverage* rather than rule coverage.

Next, consider the two sub-tables on the right, $\hat{T}^{(1)}_{\text{sub}}$ and $\hat{T}^{(2)}_{\text{sub}}$, which differ in the last attribute. Both sub-tables cover at least one rule for each tuple of $\hat{T}$. They would therefore have the same score in terms of *row coverage*. However, $\hat{T}^{(1)}_{\text{sub}}$ covers larger rules (two of size 4) compared with $\hat{T}^{(2)}_{\text{sub}}$ (only one of size 4). Accordingly, $\hat{T}^{(1)}_{\text{sub}}$ describes 28 cells of $\hat{T}$, whereas $\hat{T}^{(2)}_{\text{sub}}$ describes only 26. We therefore propose to use a *cell-based metric* rather than a row-based metric.

Finally, we note that, in $\hat{T}^{(1)}_{\text{sub}}$, if we chose row 3 instead of 1 and row 6 instead of 5 we would have the same cell coverage. However, the sub-table would be more repetitive, containing only 6-8 in the year field and two instances of medium distance. This demonstrates that coverage should be accompanied by a diversity metric, which we discuss next.

| $\hat{T}^{(3)}_{\text{sub}}$ | ROW | CANC. | DEP._TIME | SCHED._DEP. | DISTANCE |
|---|---|---|---|---|---|
| | 1 | 1 | N/A | afternoon | short |
| | 5 | 0 | morning | morning | medium |
| | 7 | 0 | evening | evening | long |

Figure 4: Example of a diverse sub-table

**Diversity.** As the above example demonstrates, diversity in sub-tables can make them less repetitive and more informative. Out of the various diversity metrics from previous work (e.g., [14, 31, 52, 64]), we adopt a standard diversity metric based on pairwise Jaccard similarity. Our particular variation of this metric, defined formally below, leverages the binning of data: each bin represents a significant subset of the data (e.g., a peak of data frequency in our implementation). Hence, we consider values from the same bin as similar and from different bins as different. As a result, intuitively, higher scores will be assigned to sub-tables that include (1) representative values from more bins; and (2) fewer repetitions per bin. Moreover, our metric accounts for values that do not participate in ARs, and for overlapping ARs, which may not be reflected in the cell coverage metric.

**Definition 2.7** (Diversity metric)**.** *The* similarity *of two tuples $t, t' \in T_{sub}$ is the ratio of cells that share a bin, formally,*
$\text{Jaccard}(t, t', T_{sub}) := \frac{\left|\{u_i \in U_{sub} | \exists B^i_j \in \mathcal{B}(u_i).\ t(u_i), t'(u_i) \in B\}\right|}{|U_{sub}|}$
*We then define the diversity of $T_{sub}$ as the complement of the average similarity between its tuples, namely,*

$$\text{divers}(T_{sub}, \mathcal{B}) := 1 - \text{avg}_{t,t' \in T_{sub}} \text{Jaccard}(t, t', T_{sub}) \quad (2)$$

**Example 2.8.** *Consider again sub-table $\hat{T}^{(1)}_{sub}$ from Figure 3. In this example, the only value repetitions are in* CANC. *and* MONTH*, yielding a diversity* $\text{divers}\left(\hat{T}^{(1)}_{sub}, \mathcal{B}\right) = 1 - \text{avg}(0.25, 0, 0.25) = 0.83$*. Figure 4 shows an even more diverse sub-table, with* $\text{divers}\left(\hat{T}^{(3)}_{sub}, \mathcal{B}\right) = 0.92$*, achieved by excluding the repetitive* MONTH *column. However, this table has lower cell coverage, describing only 24 cells, which shows there is a trade-off between the metrics.*

**Optimization problem.** We define OPT-SUB-TABLE as the problem of computing a sub-table of a predefined size that balances cell coverage and diversity. We allow users to specify target columns of interest, which will be included in the sub-table and ARs. More formally, we are given as input a table $T$ over schema $U$, dimensions $k, l$ (the number of rows and columns, respectively), a set of target columns $U^* \subseteq U$ such that $|U^*| \leq l$, a set of ARs $\mathcal{R}$ mined from $T$, a binning $\mathcal{B}$ as in Def. 2.2 and a parameter $\alpha \in [0, 1]$ which is used to balance coverage and diversity (by default, $\alpha = 0.5$). If there are target attributes ($U^* \neq \emptyset$), we retain only the rules that contain them: $\mathcal{R}^* := \{R \in \mathcal{R} \mid \{u_{r_1}, \ldots, u_{r_R+p_R}\} \cap U^* \neq \emptyset\}$. Otherwise, we retain all rules: $\mathcal{R}^* = \mathcal{R}$. Our goal is to find a $k \times l$ sub-table $T_{\text{sub}}$ that includes the target attributes ($U^* \subseteq U_{\text{sub}}$), and that maximizes the following score among all such tables:

$$\text{combined}(T_{\text{sub}}, T, \mathcal{R}^*, \alpha) = \alpha \cdot \text{cellCov}_{\mathcal{R}^*}(T, T_{\text{sub}}) + (1 - \alpha) \cdot \text{divers}(T_{\text{sub}}, \mathcal{B}) \quad (3)$$

Unfortunately, we show in the next section that directly optimizing this problem is infeasible, and therefore give in Section 4 a practical solution that indirectly accounts for ARs using table embedding.

# 3 Exact and Approximate Algorithms

We first study exact solutions for (OPT-SUB-TABLE). Unfortunately, we can already show hardness results for the cases of finding the sub-table with maximal cell coverage (MAX-CELL-COVER) and finding the sub-table with maximal diversity (MAX-DIVERSE), i.e., solving OPT-SUB-TABLE with $\alpha = 1$ and $\alpha = 0$, respectively. We therefore resort to approximate solutions. We ignore w.l.o.g. the use of target columns and binning. All proofs can be found in [50].

## 3.1 Exact Algorithms

We start by studying the complexity of MAX-CELL-COVER. Given an $n \times m$ table $T$, a brute-force algorithm can theoretically traverse all $\mathrm{O}(n^k \cdot m^l)$ sub-tables of size $k \times l$ and find the one with the maximal score. While this algorithm is polynomial in the size of $T$, it is practically infeasible due to the exponential dependency of $n, m$ in $k, l$. Even for relatively small dataset and sub-table sizes, e.g., for $n = 10,000$ and $m, k, l = 5$, this means checking $8.3 \cdot 10^{17}$ sub-tables.

Since $k$ is small, we examine whether our problem is *fixed-parameter tractable*: FPT is class of *fixed-parameter tractable* problems, having a solution in time $\mathrm{O}(\mathrm{poly}(n) \cdot f(k))$ where $n$ is the input size, $f$ is a function and $k$ is a parameter. Denote by DEC-CELL-COVER the decision problem corresponding to MAX-CELL-COVER: given a table $T$ over schema $U$, dimensions $k, l$, a set of ARs $\mathcal{R}$ and a threshold $\Theta$, decide if there exists a sub-table $T_{\text{sub}}$ of size $k \times l$ whose cell coverage is $\mathrm{cellCov}_{\mathcal{R}}(T, T_{\text{sub}}) \geq \Theta$. Unfortunately, we can show that this problem is probably not in FPT, as follows.

**Proposition 3.1.** *Given an $n \times m$ table $T$ over schema $U$, and sub-table dimensions $k, l$.* DEC-CELL-COVER *is W[2]-hard*[3] *in $n = |T|$ and $k$ as a parameter, assuming $m, l = \mathrm{O}(n)$.*

The proof (omitted) uses a reduction from DOMINATING SET and assumes that $m, l = \mathrm{O}(n)$, i.e., the number of attributes is large. Otherwise, even when $m << n$, we can still show NP-hardness in $k$.

**Proposition 3.2.** DEC-CELL-COVER *is NP-hard in $k$, the number of tuples selected for the summary, even assuming the number of attributes $m = \mathrm{O}(1)$.*

**Hardness of diversity optimization.** We now focus on the second sub-problem, MAX-DIVERSE, of selecting a sub-table with maximal diversity, another particular case of OPT-SUB-TABLE with $\alpha = 0$. In previous work, it was established that a similar diversity metric in the context of diverse crowd selection is NP-hard [64]. We show a stronger hardness for MAX-DIVERSE, establishing that similarly to MAX-CELL-COVER it is also not parameter tractable in $k$. The proof is by a reduction from INDEPENDENT SET.

[3]The W-hierarchy is a hierarchy of classes defined by properties of the translation of problems into combinatorial circuits. It is known that FPT=W[0]⊆W[1]⊆W[2]⊆ ..., and conjectured that this hierarchy is strict, i.e., W[0]⊂W[1]⊂W[2]⊂ ....

**ColumnSelection** $(T, k, l, \mathcal{R})$ // Table, dimensions, ARs

1 $T^*_{\text{sub}} \leftarrow \emptyset,\ \text{cov}^* \leftarrow -1$;
2 **while** *Sample next* $U' \subseteq U$ *such that* $|U| = l$ **do**
3 $T' \leftarrow \pi_{U'} T$; // Projection of $T$ on $U'$
4 $T_{\text{sub}}, \text{cov} \leftarrow$ **GreedyRowSelection**$(T', k, \mathcal{R})$;
5 **if** $\text{cov} > \text{cov}^*$ **then** $\text{cov}^* \leftarrow \text{cov},\ T^*_{\text{sub}} \leftarrow T_{\text{sub}}$;
6 **if** $\text{timeLimitReached}()$ **then break**;
7 **return** $T^*_{\text{sub}}$;

**GreedyRowSelection** $(T', k, \mathcal{R})$

8 $T^{**}_{\text{sub}} \leftarrow \emptyset,\ \text{cov}^{**} \leftarrow -1$;
9 **for** $i$ *in* $1 \ldots k$ **do**
10 $T^*_{\text{sub}} \leftarrow T^{**}_{\text{sub}},\ \text{cov}^* \leftarrow \text{cov}^{**}$;
11 **for** $t \in T' - T^*_{sub}$ **do**
12 $\text{cov} \leftarrow \text{cellCov}_{\mathcal{R}}(T, T_{\text{sub}})$;
13 **if** $\text{cov} > \text{cov}^*$ **then** $\text{cov}^* \leftarrow \text{cov},\ T^*_{\text{sub}} \leftarrow T_{\text{sub}}$;
14 $\text{cov}^{**} \leftarrow \text{cov}^*,\ T^{**}_{\text{sub}} \leftarrow T^*_{\text{sub}}$;
15 **return** $T^{**}_{\text{sub}}, \text{cov}^{**}$;

**Algorithm 1:** Column Combination Sampling Greedy Algorithm (CCSG)

**Proposition 3.3.** *Given an $n \times m$ table $T$ over schema $U$, and sub-table dimensions $k, l$.* MAX-DIVERSE *is W[1]-hard in $n = |T|$ and $k$ as a parameter, assuming $m, l = \mathrm{O}(n^2)$.*

## 3.2 Approximate Algorithms and Limitations

Given the hardness results above, we consider approximate solutions to MAX-CELL-COVER, i.e., computing sub-tables with an approximately-optimal score. Algorithm 1 (CCSG) greedily selects rows for given column combinations. It includes two functions. The ColumnSelection function enumerates over the possible column selections (line 2) and for each projects the data over the selected columns and computes a sub-table using the GreedyRowSelection function. The latter function iteratively attempts to add each row to the current sub-table (line 11), computes the resulting cell coverage score (line 12), and records the sub-table with maximal cell coverage. This is repeated $k$ times to select $k$ rows in total (line 9).

When no time limit is specified or when all $\binom{m}{l}$ column combinations are traversed before the time limit (line 6) is reached, CCSG achieves a cell coverage that is approximately optimal, as stated by the following proposition.

**Proposition 3.4.** *Given a table $T$ over $U$, where $|T| = n$ and $|U| = m$ and a set $\mathcal{R}$ of ARs, assuming* CCSG *fully traverses all column combinations, the resulting $k \times l$ sub-table $T_{sub}$ such that $\text{cellCov}_{\mathcal{R}}(T, T_{sub}) \geq (1 - \frac{1}{e})\,\text{OPT}$ where $\text{OPT}$ is the score of the optimal solution to* MAX-CELL-COVER. *Computation time is $\mathrm{O}(\binom{m}{l} \times k \times n^2 \times m \times |\mathcal{R}|)$.*

Note that we can get finer-grained complexity by considering the number of bins: we can remove duplicated rows in the projection on selected columns in line 3. Cell coverage can still be computed, by recording the number of copies per row. In this case, the $n^2$ component of the complexity bound above, corresponding to computing the cell coverage for each candidate row (line 12), is at most $b^{2l}$, where $b$ is the maximal number of bins per column, and thus $b^l$ is the maximal number of unique value combinations.

We perform such a duplicate elimination in our implementation. Also note that CCSG accounts for cell coverage, but not for diversity.

**Limitations.** Several practical issues arise when considering an implementation of CCSG:

First, the selection of rows via **GreedyRowSelection**, which is of quadratic complexity in the number of (unique) rows, may be too time-consuming for an interactive setting. E.g., in our experimental study, it took over 30 seconds on average to execute over a single $U'$. We therefore consider a Row and Column Combination Sampling Greedy variant (RCCSG), which is identical to CCSG, except that we further sample the rows, i.e., replace line 3 by $T' \leftarrow \mathrm{RowSample}\big(\pi_{U'}T, \mathrm{percent}\big)$. This variant is used in our experiments (Section 5), and has no approximation guarantees due to the sampling it performs.

Second, CCSG/RCCSG require ARs as input. For large datasets, even efficient implementations of AR mining, such as [13], can be too time-consuming for an interactive setting, and take several minutes to complete.

Third, CCSG/RCCSG may not be able to traverse all $\binom{m}{l}$ column combinations within the time limit required in an interactive setting. E.g., in our experimental study, neither CCSG nor RCCSG finished the column combination traversal over any dataset in less than two minutes. In this case, approximation guarantees do not hold for CCSG, and we show (Section 5) that enumerating a fraction of the column combinations yields cell coverage that is significantly lower.

# 4 Practical solution

Due to the limitations of directly optimizing our metrics, described above, we propose a different approach, which implicitly accounts for ARs using *table embeddings*. The embedding captures bin co-occurrences, and therefore, as we show below, corresponds to frequent itemsets and ARs.

## 4.1 SubTab

Our solution, shown in Algorithm 2, includes two steps: (1) Pre-Processing, which computes a vector representation for each cell in the full table $T$ using *table embedding*, and (2) Centroid-based Sub-table Selection, which uses the embedded vectors to quickly select a sub-table. Importantly, Pre-Processing is performed only once upon loading table $T$, while Selection is performed for each query that the analyst performs over $T$, to show a sub-table of the query results.

**Pre-Processing.** Given a raw table $T$, the first step (line 1 of Algorithm 2) is to normalize the values (e.g., remove illegal characters) and bin continuous columns so that values are replaced by their bin name (e.g., binning splits the *DISTANCE* column into short, medium, and long distances). Let $\tilde{T}$ be the normalized, binned table.

We then use a *table embedding process* to generate a real-valued vector for each cell in the table $\tilde{T}$. Several recent works suggested means for embedding tabular data, e.g., based on graph representations and auto-encoders [20, 21, 28, 59, 66]. Our algorithm uses the embedding as an opaque box (line 2), so any of these solutions can be plugged into it. The main embedding used in our implementation is based on [20], which is a general-purpose solution suitable for an interactive setting; we compare the results to

**Pre-processing** $(\tilde{T})$ // Raw table

1 $T \leftarrow$ normalize and bin $\tilde{T}$;

2 $\mathcal{M} \leftarrow \text{embedding}(T)$ ; // Embedding computation

3 **return** $T, \mathcal{M}$ ; // cell-to-vector model: $\mathcal{M}: T \times U \rightarrow \mathbb{R}^{\gamma}$

**Centroid-based Selection** $(T, k, l, Q, U^*, \mathcal{M})$

4 $\text{rowVecs}, \text{colVecs} \leftarrow$ empty dictionaries;

5 **if** $Q \neq \text{NULL}$ **then** $T \leftarrow Q(T)$;

6 $U \leftarrow$ columns of $T$;

7 **for** $t \in T$ **do**

8 $v \leftarrow \text{avg}_{\text{u} \in \text{U}}(\mathcal{M}(t(u)))$;

9 $\text{rowVecs} \leftarrow \text{rowVecs} \cup \{v \mapsto t\}$;

10 $\mathcal{C} \leftarrow \text{cluster}(\text{rowVecs}, k)$;

11 $T_{\text{sub}} \leftarrow \text{rowVecs.getValues}(\text{centroids}(\mathcal{C}))$;

12 **for** $u \in U - U^*$ **do**

13 $v \leftarrow \text{avg}_{\text{t} \in \text{T}}(\mathcal{M}(t(u)))$;

14 $\text{colVecs} \leftarrow \text{colVecs} \cup \{v \mapsto u\}$;

15 $\mathcal{C} \leftarrow \text{cluster}(\text{colVecs}, l - |U^*|)$;

16 $U_{\text{sub}} \leftarrow U^* \cup \text{colVecs.getValues}(\text{centroids}(\mathcal{C}))$;

17 $T_{\text{sub}} \leftarrow \Pi_{U_{\text{sub}}} T_{\text{sub}}$;

18 **return** $T_{\text{sub}}, U_{\text{sub}}$;

**Algorithm 2:** SubTab Algorithm for Sub-Table Selection

other embedding techniques in Section 5.2. Briefly, this embedding method transforms the table into a corpus of *sentences* in which each cell in the table $T$ represents a single word. We use two types of sentences: *tuple-sentences*, containing values in each tuple $t \in T$, and *column-sentences* that cover the values in $T(u), \forall u \in U$. We then train a fast implementation of word embedding [45] over a random sample of the sentences ($100K$ by default). The trained word-embedding model outputs a vector representation for table cells.

The output of this process (line 3 of Algorithm 2), is the binned table $T$ with $n$ rows and $m$ columns, and a mapping $\mathcal{M}$ between each cell in $T_{ij}$ to a corresponding, learned $\gamma$-dimensional *cell-vector* $\mathcal{M}_{ij}$ ($\gamma$ is typically a low dimension depending on the embedding technique and parameters).

The complexity of this step is the sum of costs of normalization, binning and embedding learning, which may depend on the data size, distribution and choice of implementation. For instance, in our implementation (see Section 5.1), normalization is linear in the data. The cost is dominated by binning, which requires ordering the values in each column in $\text{O}(mn \log n)$, but then computing the bins can be done in linear time [49]; and by embedding, which takes $\text{O}(in'm\gamma \log |V|)$, where $i$ is the number of epochs used for training, $n' \leq n$ is the size of the row sample used by the embedding function (by default we use, $n' = \max\{n, 100K\}$), and $V$ is the set of unique bin names in the sample [44, 45]. This is a major improvement compared with the complexity of CCSG, which has a quadratic dependency on $n$ as well as an $\binom{m}{l}$ component (by Prop 3.4).

**Centroid-Based Sub-table Selection.** After computing vector representations $\mathcal{M}$ for each cell in $T$, we use them to compute representations for rows and columns, which in turn are used to select rows and columns for the sub-table. As mentioned above, this

step is done over the results of each query, without re-executing Pre-Processing and, in particular, without recomputing the embedding.

Concretely, we first compute for each tuple $t \in T$ a vector representation, or *tuple-vector*, by taking the component-wise average of the vectors representing $t$'s cells. That is, we average the cell-vectors $\mathcal{M}(t(u_1)), \mathcal{M}(t(u_2)), \ldots, \mathcal{M}(u_{|U|}))$ (lines 7-9). To select a sub-table $T_{\text{sub}}$ with $k$ rows, we then cluster the tuple-vectors into $k$ clusters and use the $k$ rows corresponding to the cluster centroids. Next, to select $l$ columns for $U_{\text{sub}}$ we perform a similar process, creating *column-vectors*, forming clusters, and finding their centroids. The only change compared to row selection is due to the presence of target columns $U^* \subseteq U$, which must be included in the sub-table. We exclude these columns from clustering, compute only $l - |U^*|$ clusters and then add the $U^*$ columns to the selected centroids.

The time cost of this step includes the computation of row vectors (line 7-9) by averaging $m$ cell vectors of dimension $\gamma$ for each of the $n$ rows in time $\mathrm{O}(nm\gamma)$, and the computation of column vectors (line 12-14), again in $\mathrm{O}(nm\gamma)$. It also includes two invocations of the $\mathrm{cluster}$ function, one with $n$ row vectors and $k$ clusters (line 10) and one with $\leq m$ column vectors and $\leq l$ clusters (line 15). The cost depends on the clustering function. For instance, in our implementation (see Section 5.1) we use a K-means clustering algorithm whose time complexity is $\mathrm{O}(jkn\gamma)$ for rows, and $\mathrm{O}(jlm\gamma)$ for columns. $j$ is the number of iterations required to convergence, $k$ or $l$ is the number of clusters, $n$ or $m$ is the number of items and $\gamma$ is the size of each vector [43]. Compared with the Pre-processing step, this step is linear in the data ($j$ is typically a small constant), and hence of a lower complexity as well as much lower execution time in practice (see Section 5.2)

### 4.2 Embedding vs. Direct Optimization

We conclude with a few observations on our embedding-based approach for sub-table generation. First, note that Algorithm 2 does not directly optimize the cell-coverage and diversity metrics, so there are no formal guarantees for the scores its sub-tables achieve. Still, experiments in Section 5 show that SubTab computes high-quality sub-tables in a few seconds, whereas directly optimizing the metrics requires over *24 hours* to achieve a quality comparable to SubTab.

Intuitively, SubTab performs well since (1) the embedded vectors are learned w.r.t. the frequency of values co-occurrences, similarly to ARs, therefore obtains high cell-coverage; and (2) the algorithm selects cluster centroids as rows and columns in the sub-table, and therefore obtains *high diversity scores* as vectors in different clusters are significantly less similar than those in the same cluster.

To confirm the latter intuition, we measured the overlap of ARs between the row clusters. Figure 5 shows these results on the Flights dataset. Each cluster corresponds to a row in the resulting sub-table, and the heatmap shows the Jaccard similarity of the ARs for every pair of clusters. Observe that for most of the pairs, the overlap is negligible. (Similar results were obtained for the other datasets.)

## 5 Experiments

We performed an extensive experimental evaluation of SubTab both in terms of the running time as well as the quality and usefulness of computed sub-tables. After describing the experimental setup (Section 5.1), we report our results.

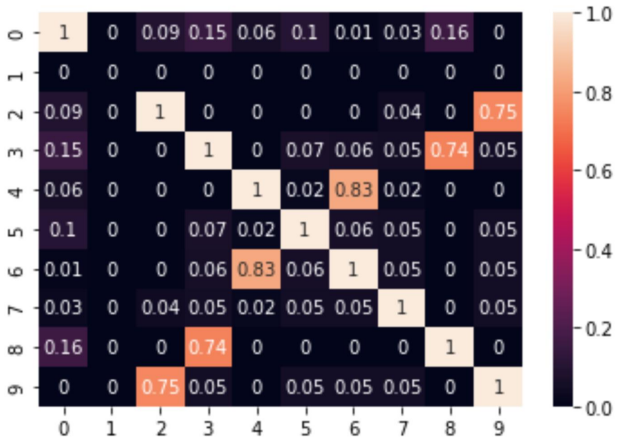



Figure 5: Heatmap showing rule overlap in row clusters.

The first set of experiments (Section 5.2) measure the performance of SubTab for each dataset using different baselines. The experiments are split into two parts: baselines that run in interactive time (up to 2 minutes), for which the comparison is based on our quality metrics; and slower baselines for which the comparison is based on both running time and quality.

The second set of experiments tests the quality and usefulness of computed sub-tables (Section 5.3) through (1) a twofold user study, where participants are asked to perform a realistic data analysis task and we evaluate their objective performance as well as usability rantings they provided for sub-tables; and (2) an offline, simulation-based study, in which we replay pre-recorded real-life analysis sessions, generate a sub-table for each exploratory query, then check whether fragments of the next query in the session (e.g., selection terms or aggregation columns) appear in the sub-table. We also use these results to directly test the correlation between the quality metrics (cell coverage, diversity and the combined score) of a given sub-table and its ability to predict the fragments of the next query. Finally, the performance using different parameter settings is tested in Section 5.4. A summary of findings from the experiments can be found in Section 5.5.

## 5.1 Experimental Setup

SubTab is implemented in Python 3.8 as a local Python library that hooks into Pandas [61] and therefore can be used, e.g. in common EDA environments such as Jupyter notebooks. The binning method used is based on kernel density estimation and is implemented with *sciPy* [10]. The Word2Vec embedding method is implemented by *gensim* [7]. Centroid selection is performed by creating clusters via KMeans using *sklearn* [9]. The experiments were run on an Intel Xeon CPU-based server with 24 cores and 96 GB of RAM.

**Metrics implementation.** Although SubTab does not require the computation of ARs, we mine ARs for experiments that measure cell coverage. If ARs are given, our implementation also supports highlighting ARs (as shown in Figure 2), which gives additional intuition to the users. We measure the effect of highlighting ARs in all of the baselines in Section 5.3.1. ARs are mined using *efficient-apriori* [4]; by default, we set

the thresholds for support and confidence to $0.1$ and $0.6$, respectively, and the minimum AR size to 3. We vary these parameters in Section 5.4. When target columns are specified, the data is split and mined separately according to the binned values of the target columns. Intuitively, this allows our metrics to reflect ARs for different target values (e.g., *CANCELLED*=0 or 1). Lastly, in our combined score, we assign equal weights to cell coverage and diversity ($\alpha = 0.5$).

**Datasets.** We demonstrate the performance of SubTab over datasets in different domains; for lack of space, we show results only for representative datasets in each experiment.

- Flights ($FL$) [2], 6M rows $\times$ 32 columns
- Spotify ($SP$) [11], 42K rows $\times$ 15 columns
- Cyber-security ($CY$) [3], 30K rows $\times$ 15 columns
- Credit card frauds ($CC$) [1], 250K rows $\times$ 31 columns
- US Funds ($USF$) [5], 23.5K rows $\times$ 298 columns
- Bank Loans dataset (BL) [6], 110K rows $\times$ 19 columns

**Baselines.** some of the baselines we have tested implement (a restricted version of) one of our algorithms. Others use different forms of iterative sampling, and get as a parameter a time limit after which they are halted.

1. SubTab: our implementation for Algorithm 2 using, for cell embedding, the fast Word2Vec implementation of [45] as described in Section 4.1.

2. EmbDI: our implementation for Algorithm 2 using, for cell embedding, the approach of [21]. Briefly, the table is transformed into a graph capturing relationships between cells, rows, and columns, and Node2Vec is used to compute vector representations. We chose EmbDI over other table embedding techniques that have recently been proposed, as it was shown to perform well on common database tasks.

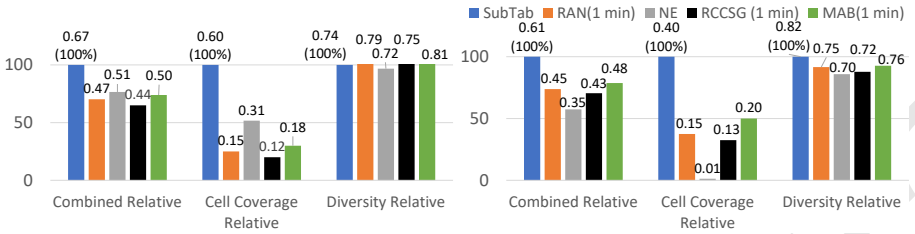
3. No embedding (NE): a restricted variant of SubTabexcluding the vector embedding step. Instead, using one-hot encoding [8], we transform categorical and textual values to numerical values and then cluster the rows and columns as in Algorithm 2. This approach corresponds to taking the first level of the hierarchical clustering in [40] (see the detailed comparison in Section 6).

4. Random (RAN($\cdot$)): sample $k$ rows and $l$ columns uniformly at random. To increase the quality of this baseline, we repeat the random selection for a user-specified amount of time and return the sub-table with the highest combined score among all the randomly computed sub-tables.

5. Multi-Armed Bandit (MAB($\cdot$)): an improved sampling-based technique using a version of the Multi-Armed Bandit algorithm [54]. The algorithm iteratively selects a sub-table and rewards its rows and columns based on the combined score. We use Upper Confidence Bound (UCB) [38] to balance exploration (examining new rows/columns) and exploitation (selecting high-reward rows/columns).

6. Column Combination Sampling Greedy (CCSG($\cdot$)): as defined in Section 3.2.

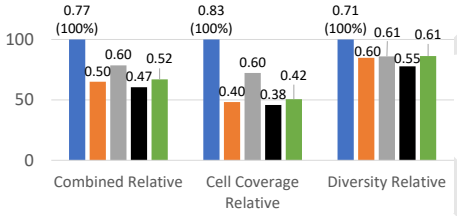
7. Row and Column Combination Sampling Greedy (RCCSG($\cdot$)): as defined in Section 3.2.




(a) SP

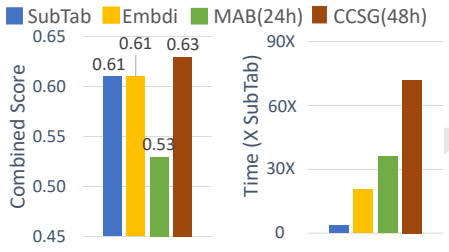



(b) FL

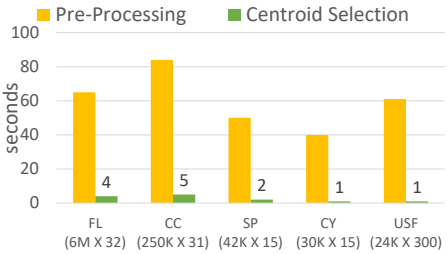



(c) CY

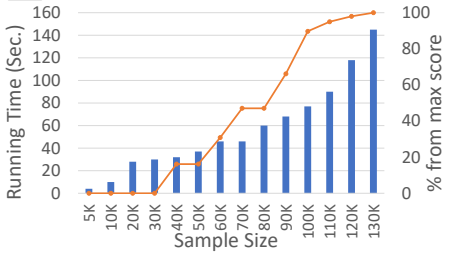
Figure 6: Experimental results for the quality of sub-tables computed by different baselines for different datasets




(a) Quality and execution time comparison with non-interactive baselines




(b) Average running times of SubTab steps: Pre-Processing and Selection




(c) Running times (bars) and combined score (curve) for different sample sizes ($FL$)

Figure 7: Experimental results for execution times and performance

## 5.2 Experimental Results

We next compare the performance of different baselines. We split the results into two parts: first, we consider baselines which run in *interactive time* (up to 2 minutes): NE, RAN(1min), MAB(1min), RCCSG(1min) and SubTab. For these algorithms, we compare output quality. Next, we consider slower algorithms (execution time 30m-2d for the datasets that we examined), and compare them to SubTab in terms of both quality and execution time. These include EmbDI and longer executions of sampling-based algorithms.

**Interactive-time baselines.** Figure 6 shows our combined score, cell coverage and diversity over the $FL$, $SP$ and $CY$ datasets. For all three datasets, SubTab achieves the highest cell coverage and combined scores, by up to $40\%$ over the other baselines. This shows that vector embedding is useful in capturing ARs (compared to NE), and that sampling for a short time is not sufficient for finding high-coverage sub-tables, even with the improvements of MAB or RCCSG. Interestingly, in the $FL$ and $CY$ datasets, SubTab also achieves the highest diversity score, which means that it outperforms the baselines for any choice of $\alpha$. In the $SP$ dataset, RAN and MAB have a slightly better diversity, but their combined score is lower due to significantly lower cell coverage score.

**Non-interactive baselines.** To gain further perspective on the quality of sub-tables, we also compare SubTab to the slower, non-interactive baselines. Figure 7a shows the results for the $FL$ dataset, left compares their performance in terms of combined score, and Figure 7a right compares the total running times. SubTab achieves the same combined score as the EmbDI baseline; however, the latter takes 40 minutes to execute, whereas SubTab requires only about 1 minute (measured end-to-end, including pre-processing and sub-table selection). This shows that the embedding method used by default in SubTab is already as effective for our purpose as state-of-the-art embedding methods and hence is preferable due to faster response times. MAB(24h) achieves the lowest quality score, even though it is executed for a long time. Finally, the Greedy(48h) baseline slightly outperforms the others in terms of quality but is the slowest – this score was achieved by executing it for 48 hours on a multi-process architecture. These results empirically justify using SubTab as an alternative to directly optimizing the score, since it can compute sub-tables with similar quality scores in only a fraction of the time. We also examined the performance of these baselines on the other datasets, as well as on samples of the $FL$ dataset with as few as 5K rows. In all of these experiments, including the 5K sample, the other baselines exceeded the time limit of 2 minutes. The observed trends for both comparative quality and execution times were similar for these other datasets, hence we omit them.

**Execution time analysis.** Recall that SubTab has two distinct steps: Pre-processing, which is executed once upon loading a data table, and Selection, which is executed for each SubTab display, both for the table itself and for queries over it (see Figure 1). Figure 7b shows the execution times for each step over different datasets, mentioning the size of the dataset next to the dataset 2-letter code. Pre-processing takes the longest time, 85 seconds, for the $CC$ dataset, although it is smaller than $FL$. The reason is that this data contains only numeric columns that must undergo binning. Still, this is a reasonable time for the set-up phase of an EDA session. The Selection phase takes only

a few seconds for all the datasets. We have tested the computation time for various sub-table sizes, and the results were similar (the difference is less than $10\%$). This shows that our reuse of embedding is indeed effective in achieving a fast response time. We also examine the scalability of our algorithm by isolating the effect of the sample size used to compute the embedding (see Section 4.1). Figure 7c shows the execution time for varying sample sizes (bars) as well as the combined score of the computed sub-table (curve). Execution time increases roughly linearly with the sample size, whereas the combined score starts to converge as the learned model approaches its maximal quality (in this case, around 100K).

## 5.3 Extrinsic Evaluation with Real Users

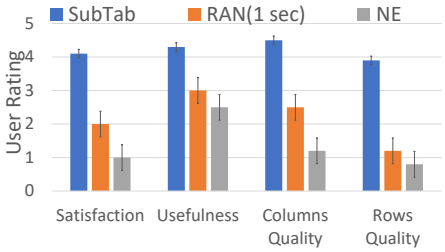
So far, our evaluation of sub-tables was done using the quality metrics we have defined. We now use extrinsic quality indicators based on EDA sessions of real users, in order to validate (1) our baseline comparison by independent experiments; and (2) the use of our intrinsic metrics by testing the correlation of diversity, cell coverage and the combined score with extrinsic usefulness metrics. The EDA sessions were obtained via a user study as well as from an existing repository [46], as described below.

### 5.3.1 Live User Study

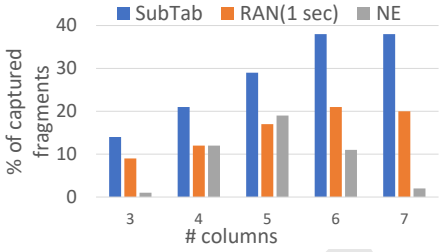
We conducted a user study where participants were asked to perform an actual data-analysis task, in which they used sub-tables to discover insights about datasets. The task was followed by a questionnaire, asking the participants to rate different aspects of the system usability. We detail the procedure below, and then provide an objective evaluation of the insights that participants discovered, as well as an analysis of the usability ranking.

We recruited 15 participants from both academic and industrial backgrounds, with varying degrees of expertise in data analysis using Pandas (from junior to expert). We divided the participants into three groups that worked with SubTab, RAN(1m) and NE, respectively, as representative baselines out of the ones that work in interactive time. The system names were hidden from participants throughout the experiment. In our previous experiment, MAB(1m) and RCCSG(1m) performed similarly to RAN(1m), and were thus excluded. Each participant used the assigned baseline to perform an exploration task on three of our datasets ($FL$, $SP$, and $BL$). The user's goal was to write down insights that were relevant to the given task while examining the sub-tables that were created during the exploration. We then counted the number of correct insights that participants listed and took the average over all exploration tasks, per participant.

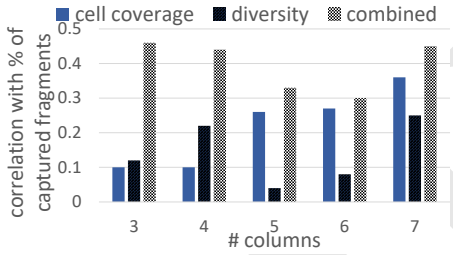
**Insights discovery.** Our first extrinsic quality measure is based on objective performance in EDA tasks. A common practice for evaluating discovery-oriented tasks is via asking the users to list insights about the data [17, 29, 51]. Here, insights correspond to ARs (e.g., flights with $X$ also have $Y$) or correlations between columns (e.g., flights with lower $X$ have higher $Y$). The correctness of insights is evaluated by measuring the support and confidence of the AR or the correlation of mentioned columns, as well as its direction (positive/negative). The reasons for using this type of open-ended task are (1) to preserve the exploratory nature of the task, and (2) to avoid biasing the exploratory search of participants, by leading them to concrete data patterns that form only a small fraction of the space of potential discoveries. For each dataset, we gave participants a notebook containing several exploratory queries, whose




(a) Questionnaire results – average user ratings with standard error




(b) Results of the collected sessions study ($CY$)




(c) Correlation between metrics in collected sessions study

Figure 8: Experimental results for extrinsic evaluation

| Metric | SubTab | RAN | NE |
|---|---|---|---|
| # correct insights | **4** (85%) | 1.2 (30%) | 0.2 (6%) |
| users with no insights | **0** | 12% | 89% |
| # Total insights | **4.5** | 3.67 | 1.5 |

Table 1: Results of the user study

results were displayed as sub-tables generated by either SubTab, RAN, or NE, unknown to the participants. The notebooks contained various exploration commands that were observed in different exploration notebooks in Kaggle. The participants were then instructed to examine each notebook and derive insights about the dataset, guided by a particular analysis task. For example, the instruction in $SP$, containing data about songs in the Spotify streaming service, was to discover *"what makes songs popular"*. The correctness of the participants' insights was then assessed by computing the correlation of the attributes included in the insight and determining if the insight was a significant association rule.

Table 1 shows the number and percentage of correct insights, the percentage of users who did not derive insights at all, and the total number of insights averaged across all users and datasets. Note that when using SubTab, users derived an average of 4 correct insights per dataset, which is 3X more than RAN and 12X more than NE. As for incorrect insights, users reached false conclusions using RAN and NE since the sub-tables produced were *misleading*. For example, the sub-tables exhibited a non-representative distribution of columns, or presented a random, false correlation between columns. Finally, observe that when using SubTab 100% of users were able to successfully finish the analysis task and derive at least one correct insight, which was not true for the other baselines.

**Questionnaire Results.** Next, we consider the subjective experience of participants. After they completed the insights discovery tasks, they were given a short questionnaire and asked to rate the quality of the sub-tables they have seen (without knowing which system produced them) on a scale of 1 (strongly disagree) to 5 (strongly agree). The questionnaire used the following statements, which use "sub-table" as a generic term but do not reveal any names of the underlying systems that produced the sub-tables.

- Q1: The presented system is better than the standard dataframe sub-table.
- Q2: Would you like to use the sub-table system in future data exploration tasks?
- Q3: The sub-tables' columns were relevant to the queries.
- Q4: The sub-tables' rows are representative and capture data patterns.

The questionnaire results are summarized in Figure 8a, which shows the average ratings and standard deviation for each question and baseline. Note that the average users' rating of SubTab is above 4 for all statements, and is significantly higher than RAN and NE. The variance of the scores, and particularly of SubTab is relatively small, indicating a broad agreement between users on the quality of the sub-tables.

Recall that our system also supports highlighting ARs, when given. We tested whether highlighting ARs changed the relative performance of the baselines by highlighting them for the $SP$ and $FL$ datasets (for *all* baselines), and used the regular (unhighlighted) tabular view for sub-tables of $BL$. We observed that the comparative performance was preserved whether or not the ARs were highlighted.

### 5.3.2 Collected Sessions Study

We also conducted an "offline" extrinsic evaluation of sub-tables using a collection of 122 pre-recorded data exploration sessions [46]. The sessions contain select, project, group-by, and sort operations over the $CY$ dataset. To evaluate the usefulness of sub-tables, we replayed each query in a session and generated a corresponding sub-table using SubTab, RAN or NE as representative, interactive-time baselines. We then examined whether the next query in each session contained a fragment (e.g., a group-by attribute, selection term, etc.) that appears in the sub-table of the previous query's results. Intuitively, the presence of next-query fragments in the sub-table indicates that the sub-table is useful in selecting the next exploration step.

The percentage of captured query fragments when varying the width (i.e., number of columns) of the sub-table from 3 to 7 (out of the 12 columns of $CY$) are shown in Figure 8b. Here again, SubTab significantly outperforms the baselines by up to $40\%$, notably improving relatively and absolutely as the width increases up to 6 columns; this apparently suffices to exhaust the contribution of columns to fragment capturing. In contrast, performance for NE deteriorates for larger tables, which indicates that its clusters are arbitrary rather than meaningful. In light of the task difficulty, SubTab's results are encouraging: even though its output is limited to small-sized sub-tables and query fragments can use any domain value (even ones not present in the table), its sub-tables still capture a significant percentage of real user query fragments.

**Correlation to intrinsic metrics.** We now examine the *direct correlation* between the extrinsic quality measure (percentage of captured fragments in each sub-table, as measured in the above experiment), and the sub-table intrinsic quality metrics (i.e., cell

coverage, diversity and the combined score). In the above experiment, we have collected about 1000 sub-tables of sizes 3 to 7, generated by different baselines, and with various cell-coverage scores, diversity scores, and combinations thereof (cell-coverage was in the range [0.2-0.83], and diversity in [0.3-0.71]).

Figure 8c shows the correlation (calculated using the Cramér's V [26] measure of association), for each of our intrinsic metrics to the percentage of captured fragments.

First, the results indicate that the combined score achieves significantly higher correlation than each of its components individually, which indicates that both components of the score are needed. Second, the relative importance of cell coverage, as reflected by its correlation score compared with diversity, increases with the sub-table size, intuitively since larger sub-tables are more likely to predict data patterns used in subsequent queries.

We also compare the ranking of baselines in our user study by the extrinsic and intrinsic metrics. The average combined score for sub-tables produced SubTab, RAN and NE in our live user study were $0.56$, $0.32$ and $0.15$ respectively, which matches the ranking of these baselines by user ratings (Figure 8a). Similarly, for the simulation-based study (Section 5.3.2), we computed the combined score for each sub-table and averaged per baseline and per sub-table size. The resulting rankings are identical (see Figure 8b), indicating that our metrics correlate with human judgments and with the usefulness of sub-tables in EDA sessions.

## 5.4 Parameter Tuning

We next test different parameter settings – the number of bins and the support and confidence thresholds. We varied each parameter while using the default value for the others, and examined the effect on the performance of SubTab and other baselines with respect to different datasets. The scores reported here are averaged over the $FL$ and $SP$ datasets. See the technical report [50] for further details.

**Number of bins.** Intuitively, increasing the number of bins makes the data more fine-grained – which results in a larger number of weaker rules (i.e., that cover fewer tuples). In such case, achieving high cell coverage is naturally more difficult as a sub-table now needs to capture more rules to cover the same number of cells. For example, from 5 bins to 10 bins we can observe a cell coverage decrease of 37% 47%, 80%, 47% for SubTab, RAN, NE and RCCSG, respectively. In contrast, the diversity score increases with the number of bins, for all baselines. This is because tuples are more likely to be different when using more bins.

**Support and confidence threshold.** AR support [13] reflects the ratio of tuples to which an AR applies. Setting a minimum support threshold of 0.1 means that we consider only rules that apply to $\geq 10\%$ of the data. Increasing the support threshold (from 0.1 to 0.3) leads to a minor decrease in cell coverage for SubTab, but a much higher decrease for other baselines (a decrease of 45% on average). This indicates that SubTab covers the meaningful, more significant rules, and therefore it is more resilient to the decrease in the number of rules that follows an increase in the support threshold.

The *confidence* of an AR measures the strength of the connection between its parts, i.e., the ratio, among all tuples for which the left-hand-side holds, of tuples where the right-hand-side also holds. Increasing the confidence threshold (from 0.5 to 0.8) leads

to trends similar to increasing the support threshold: cell coverage decreases by 15% 40%, 50%, 42% for SubTab, RAN, RCCSG, and MAB, respectively.

### 5.5 Summary of Findings

Results of these experiments show that sub-tables computed by SubTab exceed the quality of those computed by other interactive-time algorithms, and are comparable to time-consuming algorithms that directly optimize our metrics or use expensive state-of-the-art embedding methods. Our user study, over both live and pre-recorded EDA sessions, shows higher scores for SubTab sub-tables for both objective performance, i.e., the likelihood of discovering data patterns and useful columns/values for further analysis, and for subjective usability ratings. Finally, the results indicate that our metrics of sub-table quality are sound, robust and correlated with extrinsic sub-table quality metrics.

## 6 Related work

Four lines of work are related to our problem:

*(1) Row Sampling.* The task of sampling or selecting representative *rows* from a large dataset has been studied in previous work for several different use cases. For example, work in *Approximate Query Processing (AQP)* suggests using stratified sampling [12] and dynamic sampling [12, 15] in order to reduce the number of tuples and produce faster yet inexact query results. Additional use cases are: sampling for efficient generation of data visualizations [18,48], query results diversification [41,60]. Closer to our work, [40] suggests an interactive browsing interface for query results, which utilizes a row sampling method based on hierarchical clustering of the raw data. A similar approach is used by the baseline No-Embedding (NE), as described in Section 5.1, which obtains sub-optimal results compared to SubTab.

In contrast, sub-table selection requires the joint selection of rows and columns, as, e.g., certain rows better represent the values in certain columns and vice versa.

*(2) Feature Selection.* The task of reducing the number of columns in a dataset is an important step in many machine-learning processes, and is the topic of a plethora of research papers (see [22] for a survey). Roughly, these works can be categorized as filter methods, that output the Top-k features w.r.t. a given metric (e.g., Chi-Square, ANOVA and Information-Gain) [62]; as well as embedded [33] and wrapper [37] methods, which directly utilize the ML model to determine feature importance [24]. However, these works are ill-suited to our problem, since they (1) select columns only, and cannot be easily adapted to also select representative rows; and (2) operate w.r.t a predefined, target column and a prediction task, which may not exist in the data exploration phase.

*(3) Data summarization* Another line of work attempts to derive compressed forms of the data or produce a high-level, compact summary of the dataset. These include dimensionality reduction [27] techniques, data sketches [25] for online streams, and techniques for aggregation-focused AQP [34, 65]. In particular, [23] generates a summary using association rules in the data, yet directly mines the rules, which is too costly in an interactive exploration. In contrast, SubTab efficiently generates a sub-table which consists only of values from the original table.

*(4) Data exploration & Discovery Tools.* A plethora of work attempts to facilitate data exploration for users with varying degrees of expertise. E.g., for non-programmer

users, works such as [16, 19, 53, 57] suggest simplified exploration interfaces that allow wrangling the data without explicitly writing queries.

Furthermore, numerous tools have been devised to facilitate data and insight discovery. First, visualization recommender systems, such as Voyager [63], DeepEye [42], and LUX [39] automatically generate suitable, interesting visualizations for an input dataset. Other solutions, like TopK-Insight [58] and QuickInsights [30] produce aggregative insights and patterns from the given data (e.g., rising or falling trends).

Both types of solutions indeed surface useful visualizations and insights from the data, yet their output is highly specific: each insight or visualization depicts a single pattern, mined from one or two columns. SubTab assists the analysts in a different way, and can work effectively alongside such systems. SubTab reduces the (often very large) query results into a compact, representative sample. The resulting sub-table contains a subset of the actual raw data that captures multiple, diverse data patterns and characterizes the table as a whole.

# 7 Conclusion and Future Work

This paper presents SubTab, a framework for creating small, informative sub-tables for raw data display, a highly frequent operation in data exploration. Given a large input table, SubTab creates a sub-table with a small subset of the rows projected over a small subset of the columns. The rows and columns are chosen as representatives of prominent data patterns within and across columns in the input table. SubTab can also be used for displaying query results, enabling the user to quickly understand results and determine subsequent queries.

Directions for future research include exploring alternative data patterns (e.g., quantitative association rules); speeding up sub-table computation for queries over multiple tables (e.g., using join) or that change the table structure/values (e.g., aggregation, pivoting); and performing further experiments using synthetic data to study the effect of different data distributions, different types of binning etc. on the sub-tables calculated by different algorithms (e.g., a greedy approach).

# References


[1] Cyber Dataset. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

[2] Flights Dataset. https://www.kaggle.com/usdot/flight-delays?select=flights.csv.

[3] Flights Dataset. https://www.honeynet.org/challenges.

[4] Funds Dataset. https://pypi.org/project/efficient-apriori.

[5] Funds Dataset. https://www.kaggle.com/stefanoleone992/mutual-funds-and-etfs?select=MutualFunds.csv.

[6] Funds Dataset. https://www.kaggle.com/panamby/bank-loan-status-dataset.

[7] Gensim. https://radimrehurek.com/gensim.

[8] One hot encoding. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

[9] scikit-learn. https://scikit-learn.org.

[10] Scipy. https://scipy.org.

[11] Spotify Dataset. https://www.kaggle.com/c/bfh-spotify-challenge/data.

[12] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[13] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *PVLDB*, volume 1215, 1994.

[14] Y. Amsterdamer and O. Goldreich. Diverse user selection for opinion procurement. In *EDBT*, 2020.

[15] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003.

[16] Z. Bao, Y. Zeng, H. Jagadish, and T. W. Ling. Exploratory keyword search with interactive input. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 871–876, 2015.

[17] O. Bar El, T. Milo, and A. Somech. Automatically generating data exploration sessions using deep reinforcement learning. SIGMOD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[18] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualizaton. In *2013 IEEE International Conference on Big Data*, 2013.

[19] R. Bespinyowong, W. Chen, H. Jagadish, and Y. Ma. Exrank: An exploratory ranking interface. *PVLBD*, 9(13):1529–1532, 2016.

[20] R. Bordawekar and O. Shmueli. Exploiting latent information in relational databases via word embedding and application to degrees of disclosure. In *CIDR*, 2019.

[21] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[22] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1), 2014.

[23] J. Chen, J.-Y. Pan, C. Faloutsos, and S. Papadimitriou. Tsum: fast, principled table summarization. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*, 2013.

[24] A. Cichocki. Era of big data processing: A new approach via tensor networks and tensor decompositions. *arXiv preprint arXiv:1403.2048*, 2014.

[25] G. Cormode. Data sketching. *Commun. ACM*, 60(9), 2017.

[26] H. Cramér. *Mathematical methods of statistics*, volume 43. Princeton university press, 1999.

[27] J. P. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *J. Mach. Learn. Res.*, 16(1), 2015.

[28] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. Turl: Table understanding through representation learning. *arXiv preprint arXiv:2006.14806*, 2020.

[29] D. Deutch, A. Gilad, T. Milo, A. Mualem, and A. Somech. Fedex: An explainability framework for data exploration steps. *arXiv preprint arXiv:2209.06260*, 2022.

[30] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 International Conference on Management of Data*, pages 317–332, 2019.

[31] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Rec.*, 39(1), 2010.

[32] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM TODS*, 28(2), 2003.

[33] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar), 2003.

[34] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The control project. *Computer*, 32(8), 1999.

[35] H. Huang, Q. Yan, W. Lu, H. Lin, Y. Gao, and L. Chen. Leri: Local exploration for rare-category identification. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1761–1772, 2019.

[36] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, 2004.

[37] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *ICML*, 1994.

[38] T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[39] D. J.-L. Lee, D. Tang, K. Agarwal, T. Boonmark, C. Chen, J. Kang, U. Mukhopadhyay, J. Song, M. Yong, M. A. Hearst, et al. Lux: always-on visualization recommendations for exploratory dataframe workflows. *PVLDB*, 15(3):727–738, 2021.

[40] B. Liu and H. V. Jagadish. Using trees to depict a forest. *Proceedings of the VLDB Endowment*, 2(1):133–144, 2009.

[41] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1), 2009.

[42] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. ICDE, 2018.

[43] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge U. Press, 2008.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.

[45] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, 2013.

[46] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *KDD*, 2018.

[47] E. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE TKDE*, 15(1), 2003.

[48] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, 2016.

[49] V. C. Raykar, R. Duraiswami, and L. H. Zhao. Fast computation of kernel estimators. *J. Comp. Graph. Stat.*, 19(1), 2010.

[50] K. Razmadze, Y. Amsterdamer, A. Somech, S. B. Davidson, and T. Milo. Selecting sub-tables for data exploration. *CoRR*, abs/2203.02754, 2022.

[51] P. Saraiya, C. North, and K. Duca. An evaluation of microarray visualization tools for biological insight. In *IEEE Symposium on Information Visualization*, pages 1–8. IEEE, 2004.

[52] M. Seleznova, B. Omidvar-Tehrani, S. Amer-Yahia, and E. Simon. Guided exploration of user groups. *PVLDB*, 13(9):1469–1482, 2020.

[53] M. Singh, M. J. Cafarella, and H. Jagadish. Dbexplorer: Exploratory search in databases. *EDBT*, 2016.

[54] A. Slivkins. Introduction to multi-armed bandits. *CoRR*, abs/1904.07272, 2019.

[55] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *SIGMOD*, 1996.

[56] R. Srikant and R. Agrawal. Mining generalized association rules. *Future Gener. Comput. Syst.*, 13(2), 1997.

[57] A. Srinivasan, S. M. Drucker, A. Endert, and J. Stasko. Augmenting visualizations with interactive data facts to facilitate interpretation and communication. *IEEE transactions on visualization and computer graphics*, 25(1):672–681, 2018.

[58] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1509–1524, 2017.

[59] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. Rpt: relational pre-trained transformer is almost all you need towards democratizing data preparation. *arXiv preprint arXiv:2012.02469*, 2020.

[60] M. R. Vieira, H. L. Razente, M. C. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras. On query result diversification. In *ICDE*, 2011.

[61] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *SciPy*, 2010.

[62] D. Wlodzislaw, T. Wieczorek, J. Biesiada, and M. Blachnik. Comparison of feature ranking methods based on information entropy. In *IJCNN*, volume 2, 2004.

[63] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, 2016.

[64] T. Wu, L. Chen, P. Hui, C. J. Zhang, and W. Li. Hear the whole story: Towards the diversity of opinion in crowdsourcing markets. *PVLDB*, 8(5), 2015.

[65] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.

[66] L. Zhang, S. Zhang, and K. Balog. Table2Vec: Neural word and entity embeddings for table population and retrieval. In *SIGIR*, 2019.