Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks

Cong Yan*
University of Washington
congy@cs.washington.edu

Yeye He
Microsoft Research
yeyehe@microsoft.com

ABSTRACT

Data preparation is widely recognized as the most time-consuming process in modern business intelligence (BI) and machine learning (ML) projects. Automating complex data preparation steps (e.g., Pivot, Unpivot, Normalize-JSON, etc.) holds the potential to greatly improve user productivity, and has therefore become a central focus of research.

We propose a novel approach to "auto-suggest" contextualized data preparation steps, by "learning" from how data scientists would manipulate data, which are documented by data science notebooks widely available today. Specifically, we crawled over 4M Jupyter notebooks on GitHub, and replayed them step-by-step, to observe not only full input/output tables (data-frames) at each step, but also the exact data-preparation choices data scientists make that they believe are best suited to the input data (e.g., how input tables are Joined/Pivoted/Unpivoted, etc.).[1]

By essentially "logging" how data scientists interact with diverse tables, and using the resulting logs as a proxy of "ground truth", we can learn-to-recommend data preparation steps best suited to given user data, just like how search engines (Google or Bing) leverage their click-through logs to learn-to-rank documents. This data-driven and log-driven approach leverages the "collective wisdom" of data scientists embodied in the notebooks, and is shown to significantly outperform strong baselines including commercial systems in terms of accuracy.

ACM Reference Format:
Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3389738

1 INTRODUCTION

Data preparation, also known as *data wrangling*, refers to the process of preparing raw data from disparate sources into formats that are ready for business-intelligence reporting (BI) or machine-learning modeling (ML). Gartner calls data preparation "the most time-consuming step in analytics" [12]; while others report that business analysts and data scientists spend up to 80% of their time on data preparation [39, 40]. Automating data preparation has the potential to significantly improve user productivity, and democratize modern BI and ML practices for even non-technical users.

Not surprisingly, there is an increasing number of research efforts (e.g., [31, 32, 40, 50, 51, 53, 59, 72, 76, 77, 84]), and commercial systems (e.g., Trifacta [28], Microsoft Power Query [20], Paxata [19], Tableau Prep [26], Informatica Enterprise Data Prep [6]), all aimed at improving users productivity in data preparation. This trend is sometimes referred to as *"self-service" data preparation* or *smart data preparation* [12], as the goal is to enable non-technical users (e.g., in Microsoft Excel or Tableau) to be able to prepare data themselves without help from IT.

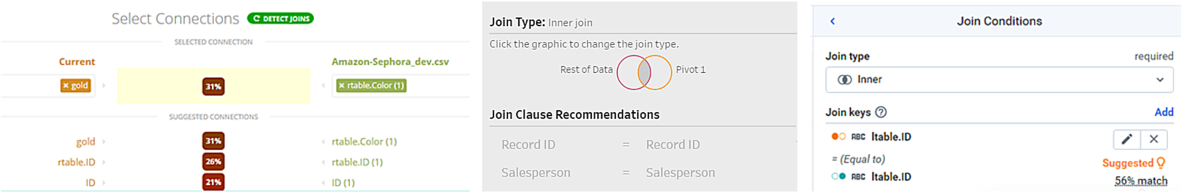
Intelligent Recommendations. Intelligent recommendation is an important class of self-service features in data preparation software, which are specifically designed to automate commonly-used operators.

As an example, for the "Join" operator, vendors like Tableau Prep [26], Paxata [19], and Trifacta [28] all have features to recommend likely Join columns. Figure 1 shows the Join-recommendation features in their respective UIs. Once users open the Join UI wizard, it gives an explicit intent to perform Join on two given tables, and these systems will recommend Join columns in a ranked list as shown in the figure, so that users only need to pick suggested Joins from the UI without needing to manually inspect input tables (which may have hundreds of columns).

*Work done at Microsoft Research.
[1] We plan to release this data set at https://github.com/congy/AutoSuggest to facilitate future research, once it is approved by an internal review.

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389738



(a) Paxata (b) Tableau Prep (c) Trifacta

Figure 1: Joins recommendation UI in commercial systems: likely join columns are suggested in ranked lists.

Similar recommendation features are also available for a few other simple operators, such as recommending GroupBy and Aggregation columns as ranked lists, when users open the GroupBy/Aggregation UI wizards.

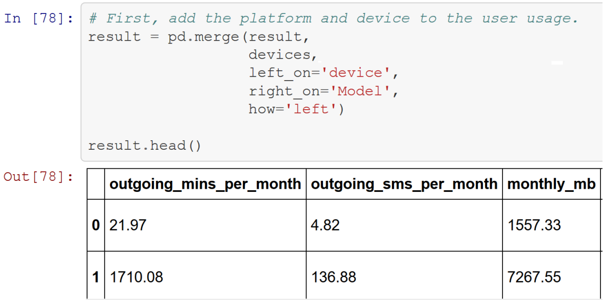
While these recommendation features are clearly beneficial, they are currently limited to operators for which simple heuristics can be devised (e.g., high value-overlap for predicting Join columns, and low-cardinality for GroupBy columns), which our analysis suggests are not always accurate.

More importantly, there are no recommendation-based features for a number of equally common but more complex operators, such as Pivot and Unpivot in the vendors we surveyed, presumably because these complex operations are more difficult to predict with simple heuristics. Given that Pivot and Unpivot are significant pain-points for users, as evidenced by a large number of questions on user-forums [3, 21, 24, 25, 29], extending intelligent recommendation to these complex operators is clearly important.

"Learn-to-recommend" with notebooks + Pandas. We in this work propose a data-driven approach to learn-to-recommend data prep operations, by leveraging a large collection data science notebooks. Specifically, computational notebooks such as Jupyter [8] are increasingly popular and have become a de-facto standard in data science. Moreover, such notebooks have become widely available in public repositories like GitHub – our crawl in Mar 2019 suggests that the number of notebooks on GitHub is around 4.7 million. Analysis shows that these notebooks cover a variety of use cases, ranging from data science projects (e.g., Kaggle), data-driven journalism (e.g., ProPublica), to reproducible academic publications.

Furthermore, we leverage the fact that Python, as well as a table manipulation API in Python called Pandas, are particularly popular in these notebooks. Pandas can roughly be thought of as a rich super-set of SQL, where some example operators are listed in Table 1.

Figure 2 shows an example step in a Python notebook. This code block calls the "merge" method in Pandas (equivalent to Join), which joins two input tables ("result" and "devices") using specified columns ("device" and "Model"), as a left-outer join. The resulting table is shown after the code block.



```
In [78]: # First, add the platform and device to the user usage.
         result = pd.merge(result,
                           devices,
                           left_on='device',
                           right_on='Model',
                           how='left')

         result.head()
```

```
Out[78]:    outgoing_mins_per_month  outgoing_sms_per_month  monthly_mb
         0  21.97                    4.82                    1557.33
         1  1710.08                  136.88                  7267.55
```

Figure 2: Example step in notebook for merge (Join).

Logical Operator	Join	Pivot	Unpivot	Groupby	Relationalize JSON
Pandas Operator	merge[17]	pivot[18]	melt[16]	groupby[14]	json_normalize[15]
#nb crawled w/ the operator	209.9K	68.9K	16.8K	364.3K	8.3K

Table 1: Popular table-manipulation operators used in the Pandas DataFrame API, and their "logical" counterparts (the entire API [13] has over 100 methods).

The fact that Jupyter notebooks and Pandas in Python are de-facto standards gives us a unique opportunity to harvest a large number of data pipelines, with real invocations of data preparation operators (Join, GroupBy, Pivot, Unpivot, etc.) on diverse data sets. We build a system to crawl, replay, and analyze such pipelines in notebooks at scale, and log detailed input/output tables (known as DataFrames in Pandas) of each operator, as well as exact choices data scientists make to manipulate tables (e.g., what columns are used in Join, how are tables Pivoted/Unpivoted, etc.)

We note that the detailed "logs" of how data scientists interact with diverse data sets is a treasure trove that allows us to learn-to-recommend data preparation steps. This is in essence analogous to the "click-through logs" used by search engines to improve search relevance.

Recommendation Tasks. In this work, we consider two types of recommendation tasks for data preparation:

• *Single-Operator Prediction*: Given input tables and a user-specified target operation (e.g. Pivot, Join, etc.), the task is to recommend suitable parameterization for the operator (e.g., how to Pivot and Join), based on characteristics of the input data. Note that the target operator is known, as the recommendation is triggered only after a user opens relevant UI Wizards (e.g., Figure 1 for Join), which gives a clear intent in terms of which operation the user wants to perform.

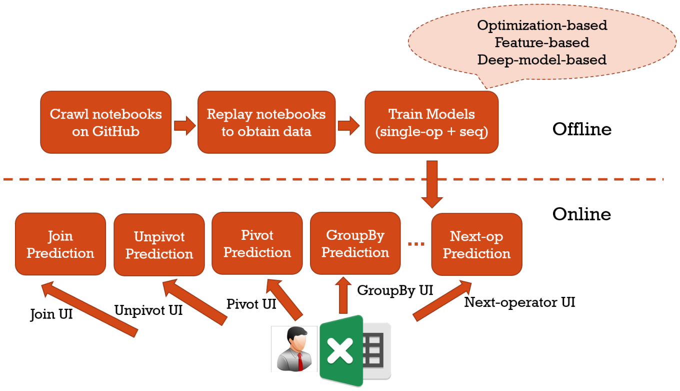


Figure 3: System Architecture.

• *Next-Operator Prediction*: Given all operations a user has performed in a pipeline up to the i-th step, predict the next operator the user would likely invoke at step $i + 1$, by exploiting latent sequential correlations between operators, as well as characteristics of input data. We note that recommending the next-operator is similar in spirit to commercial features, such as *predictive-transformation* in Trifacta [30, 53], and *smart-suggestion* in Salesforce Analytics Data Prep [23].

We develop a combination of optimization and machine-learning-based solutions for these tasks. We demonstrate that by leveraging the collective wisdom of data scientists embodied in their notebooks, our data-driven approaches are substantially more accurate than existing methods.

2 SYSTEM ARCHITECTURE

We build an end-to-end system that harvests public notebooks on GitHub to recommend data prep steps. Figure 3 shows the overall architecture of our system, which has an offline component and an online component.

At offline time, we first use the GitHub API to crawl notebooks and clone them locally. We perform syntactic analysis of the notebooks to find ones with relevant `Pandas` API calls. Statistics of collected notebooks that invoke each operator is listed in Table 1.

We then programmatically replay these notebooks and instrument their executions, to build data-flow graphs with fine-grained information of input/output tables and the operations taken at each step. We use an in-house implementation to address issues such as automatically resolving missing data-files and missing package dependency (Section 3).

Using data so collected, we build machine-learning-based and optimization-based models, to predict appropriate parameterization of each operator such as how to Pivot or Join (Section 4). We also develop deep-learning architecture to predict the next likely operator (Section 5).

At online time, the models we train can then make relevant recommendations for each operator – e.g., when users click on the Join menu button to invoke a Join UI wizard, it triggers Join recommendation that produces ranked lists of suggestions for given input tables (similar to Figure 1). Based on past operations and available tables, the next-operator can also be recommended and exposed similar to [23, 30, 53].

3 COLLECT DATA: REPLAY NOTEBOOKS

We first describe the offline component of our system that harvests and replays notebooks.

As a concrete example of the data we collect, for the notebook step in Figure 2, we would log an instance of the `merge` call (part of the `Pandas` API), together with a full dump of the two input tables “result” and “devices”, as well as additional parameters passed into the `merge` method (e.g., “device” and “Model” are join columns, and a left-outer-join is used).

We now highlight a few select aspects of our system. Additional details will be available in a full version of the paper.

3.1 Crawl Notebooks on GitHub

We leverage the GitHub API [5] to crawl notebooks based on file suffix (`.ipynb`). Statistics of notebooks with common table-manipulation operators (Join, Pivot, etc.) are shown in Table 1. The results are based on a crawl in May 2019.

3.2 Replay and Instrument Notebooks

Given a large set of crawled notebooks, we develop an end-to-end system to automatically replay the notebooks step-by-step, while collecting fine-grained information of each operator through instrumentation.

Dynamic Instrumentation. In order to replay a notebook, we use a Python tracing library [22] to instrument program execution line-by-line, which gives us access to call stacks for every function invocation, from which we extract detailed information such as parameters of function calls (including input/output tables), as well as the return value.

In the case of Figure 2, we can log all 5 parameters that are explicitly passed into this `pd.merge()` call (two of which are input tables for join), as well as 8 implicit parameters that use default values (the full `merge` API has 13 parameters[17]). Since we have a large number of notebooks to replay, we set a times-out of 5 minutes for the execution of each cell.

Handling Missing Packages. The execution of a cell can often fail for various reasons, where a common cause is missing packages in our local environment (Python library dependency is typically not explicitly specified in notebooks).

We implement a module that can parse error messages produced from the execution of a cell, to identify likely names of missing package. We programmatically install such dependencies (e.g., by invoking `pip install PKG`, where `PKG` is the name of the missing package identified from error messages). We then re-execute the failed cell, until it runs successfully or we run out of options to fix the missing dependency.

Handling Missing Data Files. Another common reason for failed execution is due to missing data files (.csv, .json, etc.), when we read data using the path specified in notebooks. This is common because notebook authors often "hard-code" absolute paths of data files in his/her local environment, as shown below:

```
df = pd.read_csv('D:\my_project\titantic.csv')
```

Such absolute paths are not valid in the GitHub repo or in our local replay environment, and will thus fail. Our replay system attempts to address missing data files in a few ways:

(1) Given a file path that we fail to load when executing a notebook (e.g., `D:\my_project\titantic.csv`), we ignore the path and search using the file name (`titantic.csv`) in the code repository, starting from the working directory;

(2) We look for URLs in comments and text cells adjacent to the failed code cell, and attempt to download missing data using the URLs extracted.

(3) Because many notebooks deal with data science challenges such as Kaggle [9], where the data sets are public and may be hosted in online data repositories. We thus also attempt to resolve missing data files by programmatically download using the Kaggle Dataset API [10] (e.g. command `kaggle datasets download -d titanic`) to download the missing dataset.

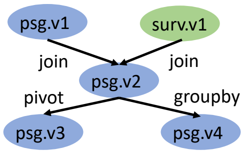
We are able to locate missing files in most cases using a combination of these methods.

3.3 Track Operator Sequences

In addition to instrumenting invocations of individual operators, we also keep track of the sequence of operations in notebooks and reconstruct the data-flow.

Specifically, we record input/output of 7 Pandas API calls that take data-frames (tables) as parameters, or produce data-frames as output. These are: `concat`, `dropna`, `fillna`, `groupby`, `melt`, `merge`, and `pivot`. We record the unique hash id of each data-frame, and trace input/output dependencies between data-frames to construct data-flow graphs (even if dependencies are far apart in the notebook).

Figure 4 shows an example of the data-flow graph for the code snippet on the right. This code snippet first reads two CSV files into data-frames, before joining the two and saving the result in `psg`. It then performs Pivot and GroupBy on `psg` for exploratory data analysis. Figure 4 shows its corresponding data-flow graph we extract, where each node is a (versioned) data-frame variable, and each edge is an operation. This allows us to construct operator sequences/pipelines, in order to predict the "next operator".



```
import pandas as pd

psg=pd.read_csv('passenger_data.csv')
surv=pd.read_csv('survive.csv')
psg=psg.merge(surv,on='PassengerId',
        how='left')
psg.pivot(header=['Survived, Pclass'],
        index='Sex', aggrfunc='count')
psg.groupby('Sex',aggrfunc='count')
```

Figure 4: Example code snippet and its data-flow.

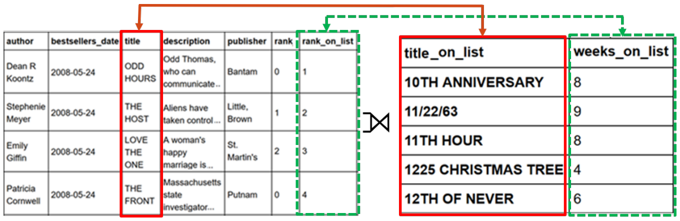
4 PREDICT SINGLE OPERATORS

Leveraging rich logs, we will first discuss "single-operator" recommendations, using Join, GroupBy, Pivot and Unpivot as example operators. Recommendation methods for additional operators such as Normalize-Json can be found in a full version of the paper.

Note that Join and GroupBy are relatively straightforward as both can be modeled as simple feature-based machine-learning. We start with the two nevertheless as they are "building blocks" required for other operators.

Pivot and Unpivot are considerably more complex – we formulate them as novel optimization problems and solve them using custom-built algorithms.

4.1 Join Predictions



author	bestsellers_date	title	description	publisher	rank	rank_on_list
Dean R Koontz	2008-05-24	ODD HOURS	Odd Thomas, who can communicate..	Bantam	0	1
Stephenie Meyer	2008-05-24	THE HOST	Aliens have taken control...	Little, Brown	1	2
Emily Giffin	2008-05-24	LOVE THE ONE	A woman's happy marriage is...	St. Martin's	2	3
Patricia Cornwell	2008-05-24	THE FRONT	Massachusetts state investigator...	Putnam	0	4

⟕

title_on_list	weeks_on_list
10TH ANNIVERSARY	8
11/22/63	9
11TH HOUR	8
1225 CHRISTMAS TREE	4
12TH OF NEVER	6

Figure 5: An example Join: The ground-truth is to join using book-titles (in solid red boxes). Existing methods using heuristics tend to incorrectly pick columns in dashed-boxes that have a higher value overlap.

Join is a widely-used operator that combines data from multiple tables. Figure 5 shows an example taken from a real notebook. The left table has a list of best-selling books, and the right one has historical information about these books. From our logs we observe that data scientists choose to left-outer-join using "title" from the left and "title_on_list" from the right (in solid boxes).

For Join we have two essential prediction tasks:

(1) Predict join columns: This is to decide which columns should be used as join keys, which is a feature available commercial systems (e.g., Figure 1), and has been studied in the literature (e.g. [36, 56, 71, 83]).

(2) Predict join types: This predicts whether the join should be inner/left-outer/right-outer/full-outer-join, etc. Since differences between these choices can be subtle and not obvious to non-expert users, accurate predictions (with intuitive explanations/visualization) would be beneficial.

Join column prediction. Given two tables T and T', with columns $\{C_1, \ldots, C_n\} \in T$ and $\{C'_1, \ldots, C'_m\} \in T'$, our problem is to find two sets of columns (S, S') that are likely join columns, with $S \subseteq T$, $S' \subseteq T'$ and $|S| = |S'|$ (note that this can be single-column or multi-columns).

We consider each pair (S, S') that are not pruned away[2] as a candidate join columns, and since we would like to produce a ranked list of candidates like in Figure 1, we model the problem as point-wise ranking [63]. Specifically, because our prediction problems use binary 0/1 labels, we use gradient boosted decision trees to directly optimize regression loss.

We use a number of features listed below:

- *Distinct-value-ratio* of S and S': Defined as the ratio of distinct tuples in S and S', over total number of rows in T and T', respectively. (In most cases at least one of S and S' should be approximate key columns with distinct-value-ratio close to 1).
- *Value-overlap* of S and S': Measured as Jaccard-similarity, as well as Jaccard-containment in both directions. Pairs with higher overlap are likely to be join columns.
- *Value-range-overlap* of S and S': If both S and S' are numeric types (e.g., numbers, date-time, etc.), we compute the min/max range of S and S', and calculate the intersection of the ranges over the union of the ranges. The intuition is that if the *range-overlap* is low, then even if we have a perfect containment in *value-overlap* (e.g., S has integers from 0-10 and S' has 0-1000), we are still not as confident whether the pair should be the Join column.
- *Col-value-types*: Column types can be string vs. integer vs. float, etc. In general two string columns with high overlap are more likely to be join columns. For integer columns the confidence is lower because there is a higher chance of two unrelated integer columns to have accidental overlap (e.g., the columns in dashed-boxes in Figure 5).
- *Left-ness*. Columns to the left of tables are more likely join columns, so we use the positions of S (resp. S') within T (resp. T'), in both absolute terms (e.g., the 2nd column from left), and relative terms (e.g., 2nd column out of 20 total cols is 2/20=10%). We use average left-ness when there is more than one column in S and S'.
- *Sorted-ness*. Whether values in S and S' are sorted (sorted columns are more likely to be key columns and Join columns).
- *Single-column-candidate*. Since we consider both single and multi-column join candidates, this feature indicates whether a candidate is single-column or not. Other things being equal, single-column joins are more likely.
- *Table-level-statistics*. These include statistics of input tables such as the number of rows in T and T', and the ratio of the two row-counts. These auxiliary features can be predictive when used in conjunction with other features – for example, if a candidate (S, S') has a high-overlap, and both T and T' have many rows, then the overlap is more trustworthy and the confidence may go up (as the chance of accidental overlap on large tables is low).

Example 1. In Figure 5, the correct join is to use "titles", despite of the fact that not all titles from one table are contained in the other (a low containment score).

Heuristics used by existing commercial systems tend to pick the incorrect pair ("rank_on_list", "weeks_on_list"), because "weeks_on_list" are fully contained in "rank_on_list", and "rank_on_list" appears to be a key. In comparison, using a combination of signals learned from notebooks (e.g., "title" is more to the left in the table, and is a string column for which value-overlap is more reliable), we can predict "titles" to be the correct Join.

In our experiments, we will detail a number of observations on these real ad-hoc join tasks (e.g., task characteristics and feature importance) that deviate substantially from the conventional wisdom in the existing literature. Overall, we find the data-driven approach combining multiple signals to be highly accurate.

Join type Prediction. Our second task is to predict join types, given tables T and T' and join-columns (S, S').

Existing commercial systems default to inner-join, which is reasonable since it is most common in practice (in the data we collect it accounts for 78% of the join cases). This however, can still be significantly improved.

For example, we observe that given two input tables, oftentimes the larger table (with more rows/columns) tends to be the "central" table of interest, and the smaller one tends to enrich the central table via Joins (similar in spirit to fact vs. dimension tables). As such, users often prefer to use outer-join to keep all rows in the central table, even if some rows do not join with the smaller table (inner join on the other hand, would remove non-joining rows).

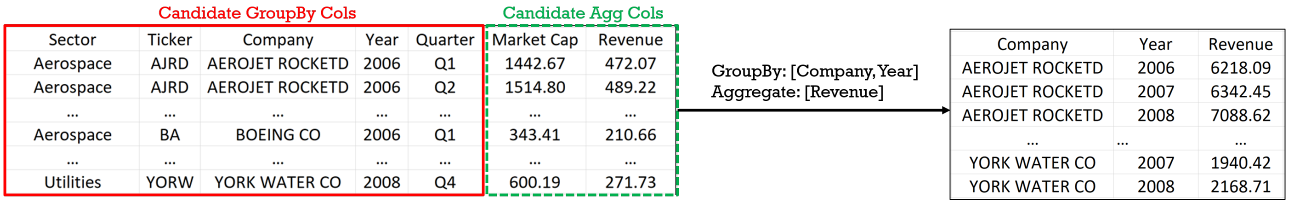
As another example, we observe that if one of the input table T' has few columns (e.g., T' may have only one column and is used as join-column), or the columns in T' are already contained in T, then the join between T and T' is likely a "filtering" step to prune down rows in T (as opposed to enriching it), where an inner-join is more likely.

We capture these subtle signals using similar features from join-column-prediction, and train point-wise ranking models. This data-driven method substantially outperforms the alternative approach of always defaulting to inner-join.

4.2 GroupBy/Aggregation

GroupBy/Aggregation are also common operators that are part of SQL. Figure 6 shows an example input table. Intuitively, we can see that any of the columns in the box ("Sector", "Ticker", etc.) can all be used as valid GroupBy columns (also known as *dimension* columns in data-warehousing terms), while columns in the dotted box ("Market Cap" and

[2] To reduce the number of candidates, we prune away obvious non-candidates based on type-mismatch (e.g., string vs. number) and sketch-based containment-checks.



Sector	Ticker	Company	Year	Quarter	Market Cap	Revenue
Aerospace	AJRD	AEROJET ROCKETD	2006	Q1	1442.67	472.07
Aerospace	AJRD	AEROJET ROCKETD	2006	Q2	1514.80	489.22
...	...	...	...	...	...	...
Aerospace	BA	BOEING CO	2006	Q1	343.41	210.66
...	...	...	...	...	...	...
Utilities	YORW	YORK WATER CO	2008	Q4	600.19	271.73

Company	Year	Revenue
AEROJET ROCKETD	2006	6218.09
AEROJET ROCKETD	2007	6342.45
AEROJET ROCKETD	2008	7088.62
...	...	...
YORK WATER CO	2007	1940.42
YORK WATER CO	2008	2168.71

Figure 6: Example GroupBy operation on an input table (left). Columns in solid box can be used as GroupBy columns (dimensions), while columns in dotted box can be used for Aggregation (measures).

"Revenue") can be used for aggregation (also known as *measures*). The right of Figure 6 shows an instance of GroupBy on "Company" and "Year" and Aggregation on "Revenue".

Prediction Task. Given a table T, and columns $\{C_i\} \in T$, our task is to make an independent prediction of whether each C_i is a reasonable choice for GroupBy and Aggregation column[3]. Predictions so produced can again be presented as ranked lists for users to select (similar to Figure 1). Because users typically have high-level ideas of desired results (e.g. "revenue by company and by year"), instead of requiring users to inspect every column (there may be hundreds of columns), they can simply inspect a list of suggested GroupBy vs. Aggregation columns to quickly produce desired results.

Like Join-prediction, we use features to describe each column $C \in T$, and feature-based models to rank likely GroupBy/Aggregation columns. The features include:

- *Distinct-value-count*: This is the number of distinct values in C. GroupBy columns typically have a small cardinality. We also divide this cardinality by the number of rows in C to get a ratio-based feature.
- *Column-data-type*: String vs. int vs. float, etc. String columns are more likely used in GroupBy, whereas float is likely for Aggregation.
- *Left-ness*: Columns to the left of a table are more likely GroupBy columns; and ones to the right are more likely to be metric columns for Aggregation. These are measured in both absolute and relative terms, like in the case of Join.
- *Emptiness*: The percentage of null values in column C. A GroupBy column typically has low emptiness.
- *Value-range*: If C is a numeric column, we compute the min-max range of its values, and the ratio of distinct value count to that range. GroupBy columns (e.g., years) tend to have small ranges.
- *Peak-frequency*: The frequency of the most common value in column C (in both absolute cell-counts and as a ratio relative to the total number of rows).
- *Column-names*: Given the name of a column C, we look it up in the training data (without this C) to see how often is this name used as GroupBy vs. Aggregation, and use the corresponding counts as features. This allows us to capture common column names for GroupBy (e.g., "company", "gender", etc.) vs. Aggregation (e.g., "profit", "revenue").

We find the predictions so produced are substantially more accurate than existing features in commercial vendors.

4.3 Pivot

Pivot/Unpivot are considerably more complex than Join/-GroupBy discussed so far. They require custom-built optimization formulations, and have not studied in the literature despite their importance. (We discuss GroupBy first nevertheless, because Pivot uses GroupBy as a building-block.)

The Pivot operator transforms a flat table into a two-dimensional table, and is a difficult step that non-expert users often struggle with [3, 21, 24, 25, 29]. Figure 7 shows an example Pivot. The input table on the left has SEC filings of companies in different sectors in the year 2006, 2007 and 2008. To better understand trends, one may create a Pivot-table shown on the right. Here the names of companies and their corresponding sectors are shown on the left, while years are on the top. With this two-dimensional report, one can better compare financial performances of the same companies over the years (by reading horizontally), and compare companies within the same sector (by reading vertically).

We note that Pivot is exceedingly common in data analytics and natively supported by database vendors [46], and end-user tools (e.g. Microsoft Excel, Tableau, Trifacta, etc.).

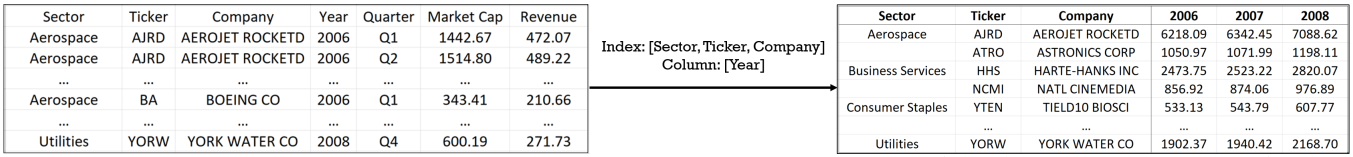
Despite its popularity, Pivot is actually quite difficult to get right – there are four required parameters to properly configure a Pivot-table: `index`, `header`, `aggregation-function`, and `aggregation-columns`. In the example of Figure 7, the `index` (left-hand-side columns of the pivot) are {"Sector", "Ticker" and "Company"}, the `header` (columns on the top of the result) is {"Years"}, the `aggregation-function` is Sum, and the `aggregation-column` is "Revenue".

Today, users are required to go through a UI-Wizard like shown in Figure 9, to properly supply these 4 parameters, which however use terminologies alien to end-users, and would typically take multiple trials-and-errors to get right.

We aim to reduce friction in creating Pivot for users, by recommending results that they likely want. This essentially requires us to accurately predict the 4 parameters in Pivot, which we will discuss below in two steps in turn.

Predict Index/header vs. Aggregation Columns. The first task here is to predict what columns are valid choices for `index`/`header` and `aggregation-column`. Intuitively, we can see that predicting `index`/`header` columns in Pivot are

[3] We do not attempt to use characteristics of T to predict aggregation-functions such as sum vs. average. Our observation is that sum vs. average are typically equally plausible in most cases – the exact choice depends more on the specific task a user has as opposed to the characteristics of T.

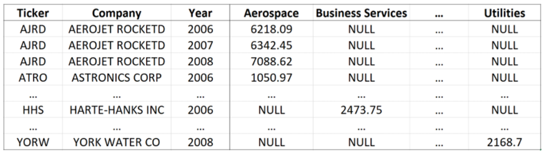


Sector	Ticker	Company	Year	Quarter	Market Cap	Revenue
Aerospace	AJRD	AEROJET ROCKETD	2006	Q1	1442.67	472.07
Aerospace	AJRD	AEROJET ROCKETD	2006	Q2	1514.80	489.22
...	...	...	...	...	...	...
Aerospace	BA	BOEING CO	2006	Q1	343.41	210.66
...	...	...	...	...	...	...
Utilities	YORW	YORK WATER CO	2008	Q4	600.19	271.73

Index: [Sector, Ticker, Company]
Column: [Year]

Sector	Ticker	Company	2006	2007	2008
Aerospace	AJRD	AEROJET ROCKETD	6218.09	6342.45	7088.62
	ATRO	ASTRONICS CORP	1050.97	1071.99	1198.11
Business Services	HHS	HARTE-HANKS INC	2473.75	2523.22	2820.07
	NCMI	NATL CINEMEDIA	856.92	874.06	976.89
Consumer Staples	YTEN	TIELD10 BIOSCI	533.13	543.79	607.77
	...	...	...	...	...
Utilities	YORW	YORK WATER CO	1902.37	1940.42	2168.70

Figure 7: Example Pivot operation that creates two-dimensional Pivot-table (right) from an input table (left).



Ticker	Company	Year	Aerospace	Business Services	...	Utilities
AJRD	AEROJET ROCKETD	2006	6218.09	NULL	...	NULL
AJRD	AEROJET ROCKETD	2007	6342.45	NULL	...	NULL
AJRD	AEROJET ROCKETD	2008	7088.62	NULL	...	NULL
ATRO	ASTRONICS CORP	2006	1050.97	NULL	...	NULL
...	...	...	...	...	...	...
HHS	HARTE-HANKS INC	2006	NULL	2473.75	...	NULL
...	...	...	...	...	...	...
YORW	YORK WATER CO	2008	NULL	NULL	...	2168.7

Figure 8: Example of a "bad" Pivot-table (with many NULLs) that uses the same dimensions as Figure 7.

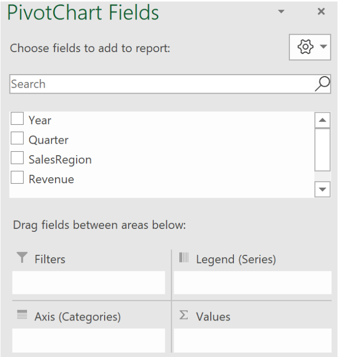


Figure 9: UI Wizard to create Pivot-table in Excel. It requires users to drag suitable columns into 4 possible buckets (shown at bottom) to properly configure Pivot, which typically takes many trials to get right. Creating Pivot in other systems is similarly complex.

essentially predicting GroupBy columns (both are *dimension* attributes); while predicting `aggregation-column` in Pivot is the same as predicting Aggregation (both are *measures*).

Example 2. Observe that the input table for Pivot in Figure 7 and the input table for GroupBy in Figure 6 are identical. Furthermore, the candidate GroupBy columns in Figure 6 (first 5 columns) are all reasonable choices as `index`/`header` in a Pivot table. Similarly, the candidate Aggregation columns in Figure 6 ("Market Cap" and "Revenue") are all valid choices for `aggregation-column` in Pivot.

We therefore directly apply the GroupBy/Aggregation prediction in Section 4.2, which would determine "Market Cap" and "Revenue" in Figure 7 as `aggregation-column`, and the rest as `index`/`header`. From here, users can pick columns of interest for the desired Pivot. In Figure 7, users would pick "Sector", "Ticker", "Company", "Year" as relevant dimensions, and "Revenue" as the `aggregation-column`.

Predict to Split Index vs. Header. From user-selected dimension columns, our second prediction task is to automatically identify a "good" placement of these columns by splitting them into `index` vs. `header`, which is difficult for users and typically require multiple trial-and-errors.

Example 3. Users have selected { Sector, Ticker, Company, Year } from Figure 7 as desired dimension columns. Since they can either be arranged as `index` (on the left of the resulting Pivot) or `header` (on the top), there are a total of $2^4 = 16$ possible choices to Pivot. Many of these arrangements are, however, not ideal.

Figure 8 shows one such example. Observe that since { Company, Ticker, Year } are selected as `index`, while { Sector } as `header`, it creates a large number of "NULL" entries in the resulting Pivot-table, because of a strong dependency between "Sector" and "Company". Splitting the two columns with one at the top and one to the left of the resulting Pivot would create a large number of empty cells (with 20 industries in the table, roughly 95% of the entries in the resulting Pivot is empty).

Similarly arranging "Company" and "Ticker" to different sides of Pivot is also undesirable as it creates even more number of empty cells.

These bad Pivots are unlikely to be selected by data scientists and in the data we collect.

We formulate the problem of splitting dimension columns into `index` vs. `header` as an optimization problem. Specifically, given columns $C = \{C_i\}$ that users select as dimensions for the desired Pivot, we need to partition them into `index` vs. `header`. This requires us to consider desirable factors such as minimizing emptiness, which we will first quantify.

Specifically, given two columns C_i, C_j, we model their "affinity score", denoted by $a(C_i, C_j)$, as the likelihood of C_i, C_j being on the same side of Pivot (both in `index` or `header`), which can be seen as their conceptual "closeness". To do so, from a large number of Pivot-tables collected from notebooks, we build a regression model to learn the affinity score between any pair of columns, using two features:

- *Emptiness-reduction-ratio*: This reduction ratio is defined as $\frac{|\{u|u\in T(C_i)\}||\{v|v\in T(C_j)\}|}{|\{(u,v)|(u,v)\in T(C_i,C_j)\}|}$, where $T(C)$ denotes values in column $C \in T$. This ratio shows how much emptiness we can "save" multiplicatively by arranging C_i and C_j on the same side. For example, Figure 8 has 20 sectors and 1000 companies, so the reduction-ratio for Sector and Company is $\frac{20*1000}{1000} = 20$, which is significant. However the reduction-ratio between Year and Sector is $\frac{3*20}{60} = 1$, indicating no saving. Attributes with higher reduction-ratio should ideally be arranged on the same side to reduce emptiness of the resulting Pivot.
- *Column-position-difference*: This is the relative difference of positions between C_i and C_j in T. What we observe is that columns that are close to each other in T are more likely

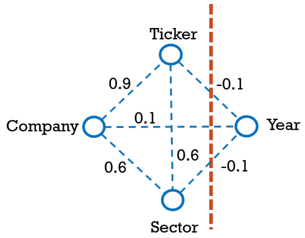


Figure 10: Example graph with affinity-scores.

to be related (e.g., "Ticker" and "Company" in Figure 7), and on the same side of Pivot.

We use all pairs of columns on the same side of real Pivot tables collected as positive examples (with affinity-scores of 1), and pairs of columns on different sides as negative examples (affinity-score of -1). We train a regression model to first predict pair-wise column affinity.

Example 4. In the example of Figure 7, we need to split dimension columns {Sector, Company, Ticker, Year} selected by users. In Figure 10, we model each column as a vertex in the graph, and use the regression model to produce affinity-scores on all edges.

We see that ("Company", "Ticker") have high affinity-score (0.9), mainly because of the high emptiness-reduction-ratio – placing the two on different sides would create 1000*1000 entries with most being NULLs, whereas placing them on the same side leads to only 1000 entries, which amounts to a 1000x emptiness-reduction.

Similarly ("Ticker", "Sector"), and ("Company", "Sector") all have high affinity-scores (0.6), because of emptiness-ratios.

Also observe that "Year" and other attributes have low affinity-scores (0.1 or -0.1), because they have low emptiness-reduction-ratios (1), and they are far apart in the table.

Given the graph with affinity-scores like in Figure 10, we can now quantify the "goodness" of an `index/header` split, and are ready to formulate it as an optimization problem.

Let C be the set of dimension columns. Intuitively, we want to partition C into C and $\overline{C}$, such that the intra-partition pairwise affinity-scores are maximized (these columns should be similar), while inter-partition pairwise affinity-scores are minimized (such columns should be dis-similar).

We write this as an optimization problem termed as AMPT (Affinity-Maximizing Pivot-Table).

$$\text{(AMPT)} \quad \max \sum_{c_i, c_j \in C} a(c_i, c_j) + \sum_{c_i, c_j \in \overline{C}} a(c_i, c_j) - \sum_{c_i \in C, c_j \in \overline{C}} a(c_i, c_j) \tag{1}$$

$$\text{s.t. } C \cup \overline{C} = C \tag{2}$$

$$C \cap \overline{C} = \emptyset \tag{3}$$

$$C \neq \emptyset, \overline{C} \neq \emptyset \tag{4}$$

The constraints in Equation (2), (3), and (4) ensure that the two partitions C and $\overline{C}$ fully covers C, are disjoint, and are non-empty, respectively.

Lemma 1. *The AMPT problem above can be solved optimally in time polynomial to the number of columns in input table T.*

It can be shown that this problem reduces to two-way graph cut [47], which can be solved in polynomial time using Stoer-Wagner algorithm [78].

Example 5. We continue with Example 4 and the graph in Figure 10. It can be shown that the best bi-section of the graph is to cut "Year" on one side, and the rest on the other side. The intra-partition affinity scores for { Company, Ticker, Sector } is (0.9+0.6+0.6) = 2.1, intra-partition affinity score for { Year } is 0 (since it is a singleton with no edge), and the inter-partition affinity score is (-0.1 - 0.1 + 0.1) = -0.1. Overall the objective function is 2.1 + 0 - (-0.1) = 2.2, which can be verified as the maximum possible on this graph.

Overall, combining an affinity-scoring model with the AMPT formulation allows us to find the most likely Pivot.

4.4 Unpivot

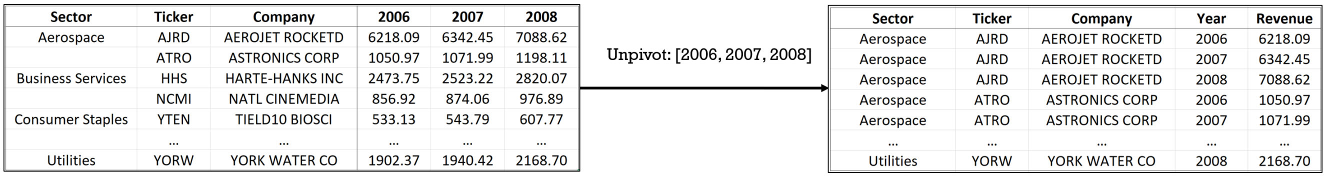
Unpivot is the inverse function of Pivot, which shapes a two-dimensional Pivot-table back to a tabular form. Conceptually, the Unpivot operation will "collapse" a selected set of columns into two new columns in the result, one being a new "key" column containing column-names of the collapsing columns, and the other being a new "value" column containing values of the collapsing columns.

Figure 11 shows such an example. Here {2006, 2007, 2008} are selected to "collapse" – in this case the column names of these 3 columns will be used to populate a new "key" column, marked as "Year" in the output; and the corresponding cell-values of these 3 columns will populate a new "value" column, marked as "Revenue" in the output. The remaining columns are left unchanged. Note that this can be seen as the inverse of the Pivot shown in Figure 7.

In Pandas API, the `melt` function implements Unpivot, where users have to specify what columns need to collapse.

Prediction Task. Our prediction task here is to predict from a table T, the set of columns that users will select to collapse in Unpivot (remaining columns are untouched).

Like in Pivot, we again compute pairwise affinity/compatibility scores for each pair of columns (C_i, C_j), using the same regression model and features discussed in Pivot. This produces a similar graph with each node corresponding to a column, and weighted edges denoting compatibility between columns. Note that to differentiate with Pivot, here we use the term "compatibility" over affinity, because in the case of Unpivot, all selected columns are collapsed into a single column, requiring them to be compatible (e.g., columns 2006, 2007, 2008 are merged into a single column in Figure 11).



Sector	Ticker	Company	2006	2007	2008
Aerospace	AJRD	AEROJET ROCKETD	6218.09	6342.45	7088.62
	ATRO	ASTRONICS CORP	1050.97	1071.99	1198.11
Business Services	HHS	HARTE-HANKS INC	2473.75	2523.22	2820.07
	NCMI	NATL CINEMEDIA	856.92	874.06	976.89
Consumer Staples	YTEN	TIELD10 BIOSCI	533.13	543.79	607.77
	...	...	...	...	...
Utilities	YORW	YORK WATER CO	1902.37	1940.42	2168.70

Unpivot: [2006, 2007, 2008]

Sector	Ticker	Company	Year	Revenue
Aerospace	AJRD	AEROJET ROCKETD	2006	6218.09
Aerospace	AJRD	AEROJET ROCKETD	2007	6342.45
Aerospace	AJRD	AEROJET ROCKETD	2008	7088.62
Aerospace	ATRO	ASTRONICS CORP	2006	1050.97
Aerospace	ATRO	ASTRONICS CORP	2007	1071.99
...	...	...	...	...
Utilities	YORW	YORK WATER CO	2008	2168.70

Figure 11: Example of an Unpivot operation that unpivots a Pivot-table (left) into tabluar format (right).

Example 6. Figure 12 shows a graph that models the case in Figure 11. Each vertex corresponds to a column in input table T, and edges weights show their pairwise compatibility.

Using compatibility scores, we again formulate the problem of finding columns to Unpivot (collapse) as an optimization problem. Unlike Pivot where the two resulting sets of columns (header and index) are symmetric, and both are required to have a strong internal affinity, in the case of Unpivot we only care about the compatibility of columns selected to collapse (e.g., 2007, 2008, 2009). The compatibility of columns not collapsing is inconsequential and need not to be high.

As such, we formulate a CMUT (Compatibility-Maximizing Unpivot-table) problem that maximizes compatibility within the group of columns selected, while minimizing the compatibility between the columns we select and ones not selected (so that we do not leave out columns highly similar to the collapsing ones in the un-selected set).

$$\text{(CMUT)} \ \max \ \underset{c_i, c_j \in C}{\text{avg}} \ a(c_i, c_j) - \underset{c_i \in C, c_j \in \mathcal{C} \setminus C}{\text{avg}} \ a(c_i, c_j) \tag{5}$$

$$\text{s.t. } C \subset \mathcal{C} \tag{6}$$

$$|C| \geq 2 \tag{7}$$

Note that compared to the objective function in Equation 1 used in AMPT, the objective function in Equation 5 of CMUT is different in two ways: (1) it has two terms as opposed to three, since it does not consider the internal affinity/compatibility of columns not selected for Unpivot; and (2) it uses average scores as opposed to sum, because sum would give undue bias towards large clusters (which is not an issue in AMPT as AMPT produces bi-sections).

We show CMUT is hard using a reduction from Densest Subgraph [34].

Theorem 2. The CMUT problem above is NP-complete.

Given the hardness of CMUT, we develop a greedy algorithm to solve it. We first select the pair of nodes with the maximum compatibility score into the target set C in CMUT, and compute the corresponding objective-function in Equation (5). In each subsequent iteration, we greedily find the node having the maximum compatibility with the current C, merge it into C, and re-compute the objective-function. The algorithm terminates when all columns are merged into C, at which point we backtrack and select the step that produces the highest objective-function value as our solution.

We use the example below to illustrate the algorithm.

Example 7. We revisit the graph in Figure 12 created from Example 6. Our greedy algorithm initializes C as the pair of nodes with the highest compatibility. Suppose the first pair picked is "2007" and "2008" (ties are broken arbitrarily). The average intra-group compatibility of this initial group is $\frac{0.9+0.9}{2} = 0.9$, and the average compatibility score of all edges crossing the cut is as $\frac{0.1*6+0.9*2}{8} = 0.3$. So the objective function for this step is $(0.9 - 0.3) = 0.6$.

We continue to the next iteration, where we find a node with the highest compatibility with the selected group, which is "2006", and merge it with "2007" and "2008". The resulting objective function can be evaluated as $\frac{0.9+0.9+0.9}{3} - \frac{0.1*6}{6} = 0.8$, which is higher than the previous iteration, and indeed a more desirable set to collapse for Unpivot.

We continue adding columns to the selected group C, but no further groups can produce a score higher than 0.8. As a result, we predict $\{2007, 2008, 2009\}$ to be the Unpivot columns.

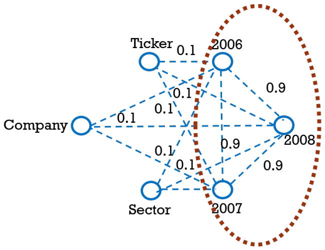


Figure 12: Example graph with compatibility-scores for Unpivot (some edges are omitted to reduce clutter).

5 PREDICT NEXT OPERATOR

So far we have focused on predicting parameters for a given target operation (Join, Pivot, etc.) that users intend to perform. In this section we describe our second overall task, which is to proactively predict the next likely operator before users provide an explicit intent. As discussed, this is similar in spirit to *predictive-transformation* in Trifacta [30, 53], and *smart-suggestion* in Salesforce Analytics Data Prep [23].

Recall that we have replayed and obtained large collections of data pipelines from notebooks, each of which is a sequence of operators, e.g., $S = (o_1, o_2, o_3, \dots, o_n)$, where each o_i is a Pandas operator (e.g., Pivot, GroupBy, Apply, etc.), invoked by users at time-stamp t_i.

The prediction task of next-operator, is to predict at time-stamp t_i, the next likely operator o_{i+1}, given operators already invoked in the past $(o_1, \dots, o_i)$, and input table T_i available at time-stamp t_i.

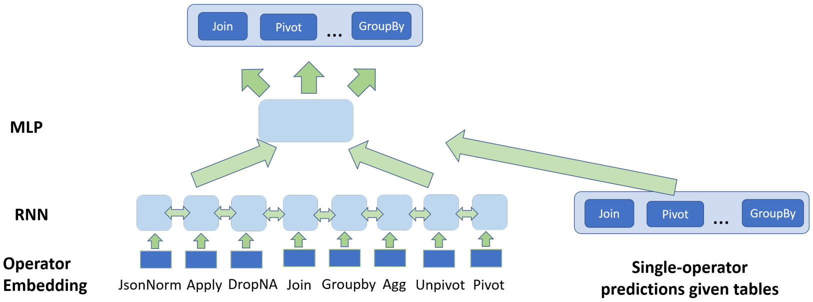


Figure 13: Model architecture to predict next operator.

Intuitively, we can leverage two main sources of signals:
(1) Since there are typically latent sequential correlations between operators (e.g., an Aggregation is likely to follow after GroupBy), we could leverage operators invoked in the past to predict the next operator;
(2) The characteristics of input tables available at time-stamp t_i are also indicative of likely operations that will follow. For example, a table T_i that "looks like" a pivot-table (e.g., Figure 11) will likely see an Unpivot invoked. We should note that such signals are implicitly captured in our single-operator models – e.g., we obtain a large objective-function value in CMUT, when T_i is appropriate for Unpivot. Thus, invoking single-operator models for each operator on T_i would utilize the characteristics of T_i to produce additional signals of whether an operator may be invoked.

The sequence-based modeling in (1) above closely resembles language-modeling problems in NLP [38], where a key task is to predict the next word given a prefix. We tested two classical approaches to this problem: an N-gram language-model from the statistical NLP literature [66], and a more recent neural approach RNN [67]. As we will report in experiments, We find RNN to be more effective in our task, which we use as the starting point of our model.

In order to also leverage characteristics of T_i at time t_i as discussed in (2) above, we invoke single-operator prediction on T_i for each operator in Section 4, and concatenate the raw scores of each operator with the continuous representation produced by the RNN layer. We note that a concatenation like this is widely used in deep models to combine information from multiple sources [48].

Figure 13 shows the resulting architecture for our model. The bottom layer on the left is an embedding layer that activates based on the presence of an operator. This layer gives a continuous representation of each operator after training. These are then fed into an RNN layer (using ReLU activation) that encodes operators invoked in the past, and produces a representation that captures the current state of the sequence at step t_i. The output of the RNN layer is then concatenated with prediction scores produced by single-operator models on T_i (shown at the bottom right of the figure). The combined vector is finally fed into an MLP layer (using Soft-max activation) to jointly produce the likelihood score of the next operator.

operator	join	pivot	unpivot	groupby	normalize JSON
#nb crawled	209.9K	68.9K	16.8K	364.3K	8.3K
#nb sampled	80K	68.9K	16.8K	80K	8.3K
#nb replayed	12.6K	16.1K	5.7K	9.6K	3.2K
#operator replayed	58.3K	79K	7.2K	70.9K	4.3K
#operator post-filtering	11.2K	7.7K	2.9K	8.9K	1.9K

Table 2: Statistics of data extracted from Notebooks.

6 EXPERIMENTS

6.1 Evaluation Datasets

We create our data set by replaying and instrumenting a large number of Jupyter notebooks on GitHub. Table 2 shows summary statistics of the data set. Because the number of notebooks with certain popular operators (e.g., Join) is too large, we sample a subset for replay in those cases.

We believe the data we collect is a representative reflection of how data scientists manipulate data in the public domain like Kaggle (there are many notebooks that we fail to replay because of missing data files, which may be proprietary enterprise data not uploaded to GitHub). We note that our approach is generic and can be deployed in proprietary domains like Enterprise Git [7], to learn from proprietary notebooks and data in these enterprises, and produce models that may be more tailored to these domains.

After a notebook is successfully replayed, we filter invocations that are deemed as duplicate (e.g., identical invocation on the same tables across notebooks, or repetitive invocations inside a loop that are similar), or uninformative (e.g., when input tables are trivially small with less than 5 rows).

The resulting data set is shown in the last line of Table 2. To the best of our knowledge this is the first systematic attempt at harvesting invocations of diverse table-manipulation operators in real pipelines, which we hope to open-source soon.

For each prediction task, we split the data 80%:20% into train and test, while making sure that examples involving the same files/data-sets are either all in train or all in test to avoid data leakage.

6.2 Methods Compared

For each prediction task, we compare the proposed AutoSuggest with two main groups of methods:

(1) Existing features available from commercial vendors, which are often strong baselines but black-box algorithms. We anonymize their names as Vendor-A, Vendor-B, etc., in accordance with their EULAs that explicitly prevent any benchmarking numbers to be revealed. We note that this is in keeping with the tradition in the database benchmarking literature [33, 42, 44, 62, 73].

(2) Related methods from the literature. These are white-box methods that we will describe separately in each task.

6.3 Experimental Setup

All experiments are performed on a Linux VM on the cloud, with 16 virtual CPU, and 64 GB memory. Auto-Suggest and alternative methods are implemented in Python 3.7.

6.4 Evaluation metric

Since most of our problems require a ranked list of suggestions, we use ranking metrics from the Information Retrieval (IR) literature [74] to evaluate suggestion quality.

Precision@K. Defined as the proportion of relevant predictions in the top-K, or $\frac{\text{\#-relevant-in-K}}{K}$. After all relevant items in ground-truth have been correctly identified, we do not penalize additional predictions at lower-ranked positions.

NDCG@K. NDCG (Normalized Discounted Cumulative Gain) is a popular metric in IR [74]. Intuitively, it computes a relevance score called DCG_K for the top-K ranked items, and compare with the score of the ideal top-K, IDCG_K. NDCG at position K is then defined as $\text{NDCG}_K = \frac{\text{DCG}_K}{\text{IDCG}_K}$, where $\text{DCG}_K = \sum_{i=1}^{K} \frac{\text{rel}_i}{\log_2(i+1)}$, in which rel_i is the relevance label of prediction at position i (in our case 0 or 1), and IDCG_K is the DCG score of the ideal ranked list at position K.

Like Precision@K, NDCG@K is in the range of [0, 1], where a higher score is more desirable.

6.5 Predict Single-Operators

We first evaluate the quality of all prediction tasks studied.

6.5.1 Predict Joins Columns.

We compare with the following methods:

- *ML-FK [71].* This is an influential approach that uses machine learning and a large number of features to discover foreign-key joins.
- *PowerPivot [36].* PowerPivot [36] employs heuristic rules to prune away unlikely join columns (e.g., boolean and numbers), and leverages content similarity to discover foreign-key joins.
- *Multi [83].* This approach leverages distributional distances between columns (e.g., EMD) to discover multi-column foreign-keys.
- *Holistic [56].* This recent approach proposes to combine distributional distances like [83], with other features.

method (all data)	prec@1	prec@2	ndcg@1	ndcg@2
Auto-Suggest	**0.89**	**0.92**	**0.89**	**0.93**
ML-FK	0.84	0.87	0.84	0.87
PowerPivot	0.31	0.44	0.31	0.48
Multi	0.33	0.4	0.33	0.41
Holistic	0.57	0.63	0.57	0.65
max-overlap	0.53	0.61	0.53	0.63
method (sampled data)	prec@1	prec@2	ndcg@1	ndcg@2
Auto-Suggest	**0.92**	-	**0.92**	-
Vendor-A	0.76	-	0.76	-
Vendor-C	0.42	-	0.42	-
Vendor-B	0.33	-	0.33	-

Table 3: Evaluation of Join column prediction. (Top) methods from the literature, evaluated on all test data. (Bottom): Comparisons with commercial systems on a random sample of 200 cases.

- *Max-Overlap.* This is a common heuristic widely used (e.g., in [39] and [36]) to predict join-columns based on value-overlap (e.g., measured in Jaccard Similarity).
- *Vendors-A/B/C.* These are commercial systems that use proprietary algorithms. Because there are no programmatic methods to test their capabilities, we report results on 200 randomly sampled cases.

We should note that most existing methods like ML-FK [71], PowerPivot [36], Multi [83], and Holistic [56] were developed specifically for foreign-key (FK) joins, and thus impose (semi)-strict checks of Uniqueness and Inclusion-Dependency. While these requirements are perfectly reasonable in a curated database setting, for the join cases we collect from data science notebooks, only 68% are strict foreign-key joins – suggesting that these joins "in the wild" are more ad-hoc than typical database joins. We thus relax the Inclusion-Dependency requirements of these FK methods when appropriate, which yields better results for these methods.

Table 3 shows a comparison of the prediction quality. On all test cases, Auto-Suggest is able to predict correctly 89% and 92% of joins at top-1 and top-2 ranked suggestions, respectively, substantially more accurate than other methods. We should note that this task is not trivial – on average 148 candidate join-columns are considered for each pair of tables, thus the low scores for some of the alternative methods.

ML-FK employs a large number of carefully engineered features, and produces strong quality results. Less sophisticated methods that use only one or two factors (e.g., only content-overlap) tend to be less accurate, suggesting that these join cases tested are likely complex, making them interesting test-beds for future research.

The bottom of Table 3 shows the comparison with commercial systems on a sample of 200 test cases, where Auto-Suggest again outperforms alternatives.

The importance scores of features used are reported in Table 4. Contrary to conventional wisdom in the foreign-key (FK) discovery literature that *value-overlap* (e.g., Jaccard

feature	left-ness	val-range-overlap	distinct-val-ratio	val-overlap
importance	**0.35**	**0.35**	**0.11**	**0.05**
feature	single-col-candidate	col-val-types	table-stats	sorted-ness
importance	**0.04**	**0.01**	**0.01**	**0.01**

Table 4: Importance of Feature Groups for Join

method	prec@1
Auto-Suggest	**0.88**
Vendor-A	0.78

Table 5: Join type prediction.

containment and similarity) may be the most important (for database FKs), we find that for ad-hoc joins performed by data-scientists in the wild, this feature group is substantially less important than many other features, such as *left-ness* and *val-range-overlap*. It is surprising to see that *val-range-overlap* is significantly more important than *val-overlap*, suggesting that containment arising from accidental overlap may be common in practice (like shown in Example 5), and thus not always a reliable signal.

6.5.2 Predict Join Types.

For this task, we compare with Vendor-A/B/C, all of which default to use inner-join as the join type (and ask users to modify if needed). We note that this is a sensible choice since it is by far the most common type of joins.

The result is shown in Table 5. Although most joins are indeed inner-joins (78% of the cases), Auto-Suggest shows a substantial improvement over the default-choice. Interestingly, we find features measuring the relative "shapes" of the two input tables (e.g., ratio of row-counts in two tables) to be most useful in predicting outer-join vs. inner-joins.

6.5.3 Predict GroupBy Columns.

This task predicts GroupBy column (Section 4.2). We compare with the following methods.

- *SQL-history [60].* SnipSuggest [60] is an influential approach that suggests likely SQL snippets based on historical queries. We adapt this to suggest GroupBy based on the frequency of columns used in the past (training) data.
- *Coarse-grained-types [68].* This approach leverages a heuristic that numerical attributes (including strings that can be parsed as numbers) are likely Aggregation columns, while categorical attributes are likely GroupBy columns.
- *Fine-grained-types [2, 65].* This approach improves upon the method above, by defining fine-grained types and assigning them as measures (Aggregation) and dimensions (GroupBy). For example, date-time and zip-code are likely for GroupBy, even if they are numbers.
- *Min-Cardinality.* This heuristic approach picks columns with low cardinality as GroupBy columns.
- *Vendors-B/C.* These are commercial systems that use proprietary algorithms.

method	prec@1	prec@2	ndcg@1	ndcg@2	full-accuracy
Auto-Suggest	**0.95**	**0.97**	**0.95**	**0.98**	**93%**
SQL-history	0.58	0.61	0.58	0.63	53%
Coarse-grained-types	0.47	0.52	0.47	0.54	46%
Fine-grained-types	0.31	0.4	0.31	0.42	38%
Min-Cardinality	0.68	0.83	0.68	0.86	68%
Vendor-B	0.56	0.71	0.56	0.75	45%
Vendor-C	0.71	0.82	0.71	0.85	67%

Table 6: GroupBy column prediction.

feature	col-type	col-name-freq	distinct-val	val-range
importance	**0.78**	**0.11**	**0.06**	**0.02**
feature	left-ness	peak-freq	empti-ness	
importance	**0.01**	**0.01**	**0.01**	

Table 7: Importance of Feature Groups for GroupBy

Table 6 shows the comparison. Prediction from Auto-Suggest is highly accurate, with a precision of 0.95 and 0.97 for the first 2 suggestions. *Min-Cardinality* performs surprisingly well, as it typically picks string-columns with low cardinality (numeric-columns tend to have high cardinality), which are often good choices. *SQL-history* also performs reasonably well, but would fail on cases where no prior SQL history can be observed. While type-based heuristics may seem reasonable, they do not work as reliably, showing the complexity of the GroupBy task.

Note that the prediction of whether *each column* is used as GroupBy vs. Aggregation, is a unit of evaluation in the result above. In order to get a big picture of the overall accuracy at the table-level (each table may have multiple GroupBy columns), we additionally report the *full-accuracy* at the table-level, which is defined as the fraction of table for which we can predict completely correctly (i.e., all GroupBy columns are ranked ahead of Aggregation columns).

This full-accuracy number is reported in the last column of Table 6. Note that we can predict GroupBy/Aggregation for 93% tables completely correctly, which is quite accurate. *Min-Cardinality* is again the second-best approach when accuracy is measured at the table-level.

The importance of features is reported in Table 7. While we expect *col-type* to be important, it is interesting to see that *col-name-freq* is the second-most important feature. Intuitively, as humans we have prior knowledge of what columns are likely GroupBy columns – e.g., "year", "department-id", etc., even if values in these columns are numbers. The *col-name-freq* feature works similarly – after seeing enough example column-names used as GroupBy in the training data, it can predict such cases accurately (e.g., columns named "year" are likely GroupBy and not Aggregation).

6.5.4 Predict Pivot: Index/header split.

For Pivot we focus on the task of splitting index vs. header columns, which we solve using an optimization formulation

AMPT (Section 4.3).[4] Since we find no recommendation features for Pivot in commercial systems, we compare with a few related methods studied in other contexts.

- *Affinity [65].* ShowMe [65] is an influential approach from the Visualization literature that studies best practices to present data based on the type of visualization. For "cross-tab" (which is similar to Pivot in spirit), an affinity heuristic is proposed to group together attributes with hierarchical relationships (e.g., FD-like attributes).
- *Type-Rules [43](Page 33, Section II).* This patent publication touches on a few simple heuristics that can be used to automatically place attributes in a pivot table based on data types (e.g., date-time, numeric attributes, etc.).
- *Min-Emptiness.* This is one of the signals considered in our AMPT, which utilizes the observation that columns with strong semantic dependency (e.g., "Ticker" and "Company") should be arranged to the same side to reduce empty cells in the resulting Pivot. We develop a greedy baseline that minimizes the fraction of empty cells (by iteratively merging pairs of columns with maximum emptiness-reduction-ratio).
- *Balanced-Split.* Since pivot-tables are often balanced in terms of width vs. height, this approach cuts given index/-header columns in a balanced manner.

Table 8 shows the quality comparison. When evaluated using full-accuracy (i.e. the split has to be completely identical to the ground-truth), our approach gets 77% of the cases correct. Both *Min-Emptiness* and *Affinity* are quite competitive, showing that minimizing empty cells is a reasonably effective approach to producing Pivot tables (which is a factor considered by AMPT). *Type-Rules* uses a static rule-based heuristics, which performs substantially worse, showing that it cannot handle diverse Pivot cases in practice.

In addition to full-accuracy, we also measure how close the predicted split is to the ground-truth. Here, we use the Rand-Index (RI) from the clustering literature [70] to evaluate result quality, where $RI = \frac{\text{\#-correct-edges}}{\text{\#-total-edges}}$, in which an edge e is deemed correct if the assignments of two vertices incident to e are the same in the prediction and the ground-truth (e.g., the two are in the same cluster or not). RI gives partial-credit to predictions that are close enough to the ground-truth, where full-accuracy only produces 0/1 scores.

We report RI numbers in the second column of Table 8, which are consistent with the full-accuracy evaluation. This again shows the benefit of AMPT that uses a principled optimization-based formulation.

6.5.5 Predict Columns to Unpivot.

For Unpivot, recall that the prediction task is to select the set of columns to "collapse" into two new columns.

method	full-accuracy	Rand-Index (RI)
Auto-Suggest	**77%**	**0.87**
Affinity	42%	0.56
Type-Rules	19%	0.55
Min-Emptiness	46%	0.70
Balanced-Cut	14%	0.55

Table 8: Pivot: splitting index/header columns.

method	full	column	column	column
	accuracy	precision	recall	F1
Auto-Suggest	**67%**	**0.93**	**0.96**	**0.94**
Pattern-similarity	21%	0.64	0.46	0.54
Col-name-similarity	27%	0.71	0.53	0.61
Data-type	44%	0.87	0.92	0.89
Contiguous-type	46%	0.80	0.83	0.81

Table 9: Unpivot: Column prediction.

We observe that input tables in the Unpivot operations we collect are typically wide, with 183 columns on average. Furthermore, 170 out of the 183 columns need to be collapsed in Unpivot on average, leaving the remaining 13 columns untouched. Given the large number of choices this presents, it is clearly a difficult prediction task.

Like Pivot, there are no recommendation-based features in the commercial systems we surveyed. There is also no existing methods in the literature that directly address the problem of predicting Unpivot. We therefore compare Auto-Suggest with a few related methods that are studied in other contexts.

- *Pattern-similarity [58].* In studying methods to restructure tables, the authors in [58] use a heuristic to Unpivot related columns, which is measured by a form of pattern similarity that they define.
- *Col-name-similarity [79].* This patent publication studies data deduplication, and proposes a few heuristics to find similar columns that can be collapsed/Unpivoted, the first of which is based on column-name similarity (measured in Jaccard). We implement it as the *col-name-similarity* baseline.
- *Data-type [79].* A second heuristic proposed in [79] uses data types (e.g., string vs. numbers) to find related columns, and is also a baseline we compare with.
- *Contiguous-type [79].* This improves on *Data-type* method above, by additionally requiring Unpivot columns to be contiguous in input table T.

Table 9 shows the comparison of prediction quality. When evaluated using full-accuracy (the full set of columns predicted for Unpivot has to be identical to ground-truth), Auto-Suggest uses the CMUT formulation and can correctly solve 67% of the cases, substantially better than other methods. While there is clearly room for improvement in the future, the fact that input tables for Unpivot have 183 columns on average makes us believe that it is a really challenging task.

We note that other methods are substantially less accurate, with *Contiguous-type* being the second-best approach.

[4] We omit details on predicting Index/header columns, as it is identical to GroupBy column prediction, and our approach has high accuracy (0.96).

We additionally evaluate the precision/recall/F1 of the columns predicted to Unpivot/Collapse, by comparing with the ground-truth. These results are shown in the last three columns of Table 9. It can be seen that over 90% of columns that we predict to Unpivot overlap with the ground-truth, suggesting that while our approach only gets 67% cases fully correct, many of the incorrect ones are mostly partially correct. From the suggested Unpivot, users may be able to use drag/drop to add/remove columns from the suggested list to quickly converge to the desired result.

6.6 Predict Next Operator

operator	groupby	join	concat	dropna	fillna	pivot	unpivot
percentage	33.3%	27.6%	12.2%	10.8%	9.6%	4.1%	2.4%

Table 10: Distribution of operators in data flows.

method	prec@1	prec@2	recall@1	recall@2
AUTO-SUGGEST	**0.72**	**0.79**	**0.72**	**0.85**
RNN	0.56	0.68	0.56	0.77
N-gram model	0.40	0.53	0.40	0.66
Single-Operators	0.32	0.41	0.32	0.50
Random	0.23	0.35	0.24	0.42

Table 11: Precision for next operator prediction.

We now describe an evaluation of the next-operator prediction task (Section 5). The distribution of operators in the crawled pipelines is shown in Table 10.

We compare results between the following methods.

- AUTO-SUGGEST. This is the proposed approach using a deep model architecture in Figure 13 (implemented using Keras [11]), which combines signals from both sequence modeling using RNN, as well as the characteristics of input tables captured by single-operator predictions (Section 4).
- RNN [67]. We also compare with a neural RNN model, which is effective for language modeling tasks in NLP (given a prefix of words, predict the next likely word). This approach uses sequence information only.
- N-gram language model. N-gram [66] is another popular language modeling approach for sequence prediction. Like RNN, this uses sequence only. We implement this using the popular NLTK [64], with trigrams and MLE estimator.
- Single-Operators. In addition to sequence-based models, we also compare with a baseline that combines predictions from all single-operator models on given table T_i. Such an approach makes predictions using only the characteristics of input tables, without considering operators invoked in the past. It provides a reference point to see how much additional benefit can be obtained by using the sequence history.

Table 11 shows the comparison. AUTO-SUGGEST clearly improves over other approaches, and can predict the next operator correctly 72% of the times at top-1, which we think is reasonable given that there are 7 possible operators in the candidate space.

Among sequence-based approaches, *RNN* is substantially more accurate than N-gram, showing its strength in modeling sequences, and is the reason we chose RNN as the starting point of our model in Figure 13. There is a sizable gap between AUTO-SUGGEST and *RNN*, showing a substantial benefit by considering the characteristics of the input table (e.g., when the input table looks like a Pivot table, the single-operator Unpivot-predictor would give a strong confidence score, boosting our next-operator prediction to be Unpivot).

Single-Operators uses only information from input tables and not the sequence, which is also less accurate, showing the need to take into account both sequences and the input tables, as is the approach we take in AUTO-SUGGEST.

7 RELATED WORKS

The research community has played a significant role in thought-leadership that has influenced the field of self-service data preparation. Prominent examples include the line of work started by Wrangler [59] and its commercial instantiation Trifacta [28]. Various methods have been proposed in the literature to automate different data preparation steps, some of which we will briefly review here.

Data transformation is a common data preparation step. Recent progress includes the use of the program-by-example paradigm, which significantly lowers the barrier to performing data transformations. Systems like FlashFill [49] and Transform-Data-by-Example (TDE) [51] allow users to provide input/output examples to specify desired transformations. Transformation programs consistent with the given examples are then synthesized using DSL [49, 76], or code on GitHub [51]. This line of work has significant impacts on commercial systems (e.g. FlashFill is available in Excel [4], TDE is used in Power BI [1, 27], etc.).

Significant progress has also been made towards automating a variety of other important operators, such as data-extraction [37, 41, 45, 61, 69], transformation-join [52, 81, 84], table restructuring [35, 57, 75], error-detection [54, 55, 80, 82], etc. Some of these advances have already influenced the commercial space and given rise to new features in existing commercial systems.

8 CONCLUSIONS

We in this work propose a data-driven approach to “learn” how data scientists manipulate diverse data sets in Jupyter notebooks, whose best-practices are then captured as predictive models to recommend data-preparation steps for less-technical users in self-service data prep software. We show the promise of such an approach, and believe that leveraging notebooks is a promising direction for future research.

REFERENCES

[1] Add-Column-From-Examples (in Power BI). http://powerbi.microsoft.com/en-us/blog/power-bi-desktop-june-feature-summary/#addColumn.

[2] Dresden Web Table Corpus. https://help.tableau.com/current/pro/desktop/en-us/datafields_typesandroles.htm.

[3] Excel Forum: Questions tagged with Pivot. https://techcommunity.microsoft.com/t5/tag/pivottable/tg-p/board-id/ExcelGeneral.

[4] FlashFill in Excel. https://www.microsoft.com/en-us/microsoft-365/blog/2012/08/09/flash-fill/.

[5] Github api. https://developer.github.com/v3/.

[6] Informatica Enterprise Data Prep. https://www.informatica.com/products/data-catalog/enterprise-data-prep.html.

[7] Informatica Enterprise Data Prep. https://github.com/enterprise.

[8] Jupyter notebooks. https://jupyter.org/.

[9] Kaggle. https://www.kaggle.com/.

[10] Kaggle dataset api. https://www.kaggle.com/docs/apiinteracting-with-datasets.

[11] Keras. https://keras.io/.

[12] Market guide for data preparation - gartner, 2017.

[13] Pandas data frame list of apis. https://pandas.pydata.org/pandas-docs/stable/reference/frame.html.

[14] Pandas groupby operator api. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html.

[15] Pandas json normalize operator api. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.io.json.json_normalize.html.

[16] Pandas melt operator api. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html.

[17] Pandas merge operator api. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.merge.html.

[18] Pandas pivot operator api. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot_table.html.

[19] Paxata data preparation. https://www.paxata.com/.

[20] Power bi. https://docs.microsoft.com/en-us/power-bi/desktop-data-types.

[21] Power BI Forum: How To Create Pivot Tables. https://community.powerbi.com/t5/Desktop/Create-Pivot-table/td-p/627586.

[22] python tracing library. https://pymotw.com/2/trace/.

[23] Salesforce Einstein Data Prep Recipe: Smart Suggestions. https://help.salesforce.com/articleView?id=bi_integrate_recipe_column_profile_suggestions.htm&type=5.

[24] StackOverflow: How To Create Pivot. https://stackoverflow.com/questions/47152691/how-to-pivot-a-dataframe.

[25] StackOverflow: Questions tagged with Pivot. https://stackoverflow.com/questions/tagged/pivot.

[26] Tableau prep. https://www.tableau.com/products/prep.

[27] Transform Data by Example (from Microsoft Office Store). https://store.office.com/en-us/app.aspx?assetid=WA104380727&ui=en-US&rs=en-US&ad=US&appredirect=false.

[28] Trifacta. https://www.trifacta.com/.

[29] Trifacta Forum: How to create Pivot. https://community.trifacta.com/s/question/0D51L0000745dOdSAI/how-to-use-pivot-table-please-provide-one-example.

[30] Trifacta: Predictive Transformations. https://docs.trifacta.com/display/SS/Overview+of+Predictive+Transformation.

[31] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 9(12), 2016.

[32] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1134–1145. IEEE, 2016.

[33] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 480–491, 2004.

[34] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1-3):15–26, 2002.

[35] D. W. Barowy, S. Gulwani, T. Hart, and B. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. *ACM SIGPLAN Notices*, 50(6):218–228, 2015.

[36] Z. Chen, V. Narasayya, and S. Chaudhuri. Fast foreign-key detection in microsoft sql server powerpivot for excel. *Proceedings of the VLDB Endowment*, 7(13):1417–1428, 2014.

[37] X. Chu, Y. He, K. Chakrabarti, and K. Ganjam. Tegra: Table extraction by global record alignment. In *SIGMOD*, 2015.

[38] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[39] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, 2002.

[40] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *Cidr*, 2017.

[41] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[42] V. Ercegovac, D. J. DeWitt, and R. Ramakrishnan. The texture benchmark: measuring performance of text queries on a relational dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 313–324, 2005.

[43] A. Folting, K. Tupaj, R. C. Collie, and A. V. Grabar. Automated placement of fields in a data summary table, us patent 7,480,675, filed by microsoft, 2009. US Patent 7,480,675.

[44] F. Funke, A. Kemper, and T. Neumann. Benchmarking hybrid oltp&olap database systems. *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2011.

[45] Y. Gao, S. Huang, and A. Parameswaran. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *Proceedings of the 2018 International Conference on Management of Data*, pages 943–958, 2018.

[46] J. Gennick. *SQL Pocket Guide: A Guide to SQL Usage.* " O'Reilly Media, Inc.", 2010.

[47] O. Goldschmidt and D. S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of operations research*, 19(1):24–37, 1994.

[48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[49] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8), 2012.

[50] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *ACM SIGPLAN Notices*, volume 46. ACM, 2011.

[51] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (TDE): an extensible search engine for data transformations. *VLDB*, 11(10), 2018.

[52] Y. He, K. Ganjam, and X. Chu. SEMA-JOIN: joining semantically-related tables using big table corpora. *VLDB*, 8(12), 2015.

[53] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[54] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International*

Conference on Management of Data, pages 829–846, 2019.

[55] Z. Huang and Y. He. Auto-Detect: Data-Driven Error Detection in Tables. In *SIGMOD*, 2018.

[56] L. Jiang and F. Naumann. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems*, pages 1–23, 2019.

[57] Z. Jin, M. R. Anderson, M. Cafarella, and H. Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 683–698. ACM, 2017.

[58] N. Kabra and Y. Saillet. Data de-duplication, 2019. US Patent 10,387,389.

[59] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[60] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for sql. *Proceedings of the VLDB Endowment*, 4(1):22–33, 2010.

[61] V. Le and S. Gulwani. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49. ACM, 2014.

[62] S. H. Lee, S. J. Kim, and W. Kim. The bord benchmark for object-relational databases. In *International Conference on Database and Expert Systems Applications*, pages 6–20. Springer, 2000.

[63] T.-Y. Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[64] E. Loper and S. Bird. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[65] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE transactions on visualization and computer graphics*, 13(6):1137–1144, 2007.

[66] C. D. Manning, C. D. Manning, and H. Schütze. *Foundations of statistical natural language processing.* MIT press, 1999.

[67] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[68] C. Ordonez. Horizontal aggregations for building tabular data sets. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 35–42, 2004.

[69] S. Ortona, G. Orsi, M. Buoncristiano, and T. Furche. Wadar: Joint wrapper and data repair. *Proceedings of the VLDB Endowment*, 8(12):1996–1999, 2015.

[70] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[71] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser. A machine learning approach to foreign key discovery. In *WebDB*, 2009.

[72] S. Schelter, D. Lange, P. Schmidt, M. Celikel, and F. Biessmann. Automating largescale data quality verification. In *VLDB*, 2018.

[73] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 974–985. Elsevier, 2002.

[74] H. Schütze, C. D. Manning, and P. Raghavan. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, page 260, 2008.

[75] A. O. Shigarov and A. A. Mikhailov. Rule-based spreadsheet data transformation from arbitrary to relational tables. *Information Systems*, 71:123–136, 2017.

[76] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *VLDB*, 9(10), 2016.

[77] R. Singh, B. Livshits, and B. Zorn. Melford: Using neural networks to find spreadsheet errors. *MSR technical report*.

[78] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[79] G. Verbruggen and L. De Raedt. Automatically wrangling spreadsheets into machine learning data formats. In *International Symposium on Intelligent Data Analysis*, pages 367–379. Springer, 2018.

[80] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 811–828, 2019.

[81] Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *SIGMOD*, 2017.

[82] J. N. Yan, O. Schulte, M. Zhang, J. Wang, and R. Cheng. Scoded: Statistical constraint oriented data error detection, 2020.

[83] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, 3(1-2):805–814, 2010.

[84] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *VLDB*, 10(10), 2017.