Teaching Data Science by Visualizing Data Table Transformations: Pandas Tutor for Python, Tidy Data Tutor for R, and SQL Tutor

Sam Lau*
lau@ucsd.edu
UC San Diego

Sean Kross*
skross@fredhutch.org
Fred Hutchinson Cancer Center

Eugene Wu
ewu@cs.columbia.edu
Columbia University

Philip J. Guo
pg@ucsd.edu
UC San Diego

ABSTRACT

Data science instructors often find it hard to explain to students how a piece of code written in Python, R, or SQL executes in order to transform tabular data. They currently resort to hand-drawing diagrams or making presentation slides to illustrate the semantics of operations such as filtering, sorting, reshaping, pivoting, grouping, and joining. These diagrams are time-consuming to create and do not synchronize with real code or data that students are learning about. In this paper we show that a step-by-step visual representation of tabular data transforms can help instructors to explain these operations. To do so, we created a table visualization library that illustrates the row-, column-, and cell-wise relationships between an operation's input and output tables. On top of this library we built a trio of free web-based visualization tools – Pandas Tutor for Python, Tidy Data Tutor for R tidyverse, and SQL Tutor – that run users' code and automatically produce diagrams of how Python/R/SQL transforms data tables step-by-step from input to output. Since launching in Dec 2021, over 61,000 people from over 160 countries have visited our website to try out these tools.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

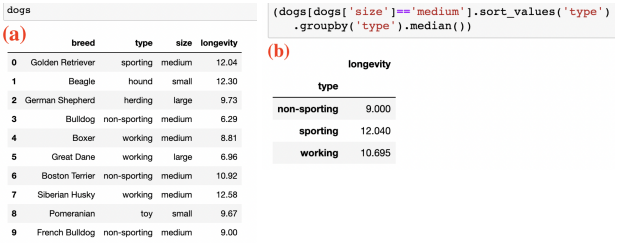
KEYWORDS

data science education; tabular data; code visualization

ACM Reference Format:
Sam Lau, Sean Kross, Eugene Wu, and Philip J. Guo. 2023. Teaching Data Science by Visualizing Data Table Transformations: Pandas Tutor for Python, Tidy Data Tutor for R, and SQL Tutor. In *2nd International Workshop on Data Systems Education (DataEd '23), June 23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3596673.3596972

1 INTRODUCTION

The past two decades have seen a sea change in the data ecosystem, and there is more demand than ever for people who have the capabilities to transform and analyze data. As such, there is a growing diversity of data-oriented roles in the workforce – such as data scientists, engineers, analysts, and enthusiasts – as well as the number of people that identify with these roles [2]. In many universities, enrollments in introductory data classes have also ballooned due to interest from across disciplines [10]. For instance, at Columbia University, *W4111 Introduction to Databases* was the 6th largest class on campus in Fall 2021; at UC San Diego, *COGS108 Data Science in Practice* is one of the largest in our department.



	breed	type	size	longevity
0	Golden Retriever	sporting	medium	12.04
1	Beagle	hound	small	12.30
2	German Shepherd	herding	large	9.73
3	Bulldog	non-sporting	medium	6.29
4	Boxer	working	medium	8.81
5	Great Dane	working	large	6.96
6	Boston Terrier	non-sporting	medium	10.92
7	Siberian Husky	working	medium	12.58
8	Pomeranian	toy	small	9.67
9	French Bulldog	non-sporting	medium	9.00

type	longevity
non-sporting	9.000
sporting	12.040
working	10.695

Figure 1: It can be hard for instructors to explain how an input table of data (a) gets transformed by Python/R/SQL code into an output table (b). We created visualization tools to help instructors teach these kinds of table transformations.

Alongside this demand, there is a growing ecosystem of tools centered around dataframe programming APIs [19] in Python and R (e.g., pandas [4], tidyverse [5]), as well as the longstanding SQL ecosystem. At their core, these different languages and APIs all share the same underlying set of relational constructs (e.g., select, project, join, union, aggregation). Yet as data science instructors, we repeatedly encounter two major challenges when teaching data manipulation using one or combinations of these popular tools:

1) Individual code statements can be hard to understand. Dataframe APIs encourage programming idioms that perform multiple operations in a single dense statement [23]. For instance, say that Alice is teaching an introductory course and writes this Python pandas statement to analyze a dataset about dogs (Figure 1a):

```
dogs[dogs['size']=='medium'].sort_values('type')
  .groupby('type').median()
```

When she runs that code, it produces the output table in Figure 1b. Yet students may find it hard to understand *how* the input table was transformed into the output, because the statement actually performs four operations: filtering, sorting, grouping by a column, and aggregating within-group medians. Note that Alice could tediously break up her code into four separate lines and try to explain the intermediate outputs of each. But that still leaves the problem that grouping and aggregation are individually hard to explain. This problem is not limited to Python. The same concept might be presented in R using the tidyverse [5] API or as a SQL query:

```
dogs %>% filter(size == "medium") %>%  # R tidyverse pipeline
  arrange(type) %>% group_by(type) %>%
  summarize(longevity = median(longevity))

SELECT median(longevity) FROM dogs WHERE size = 'medium'
  ORDER BY size GROUP BY type  # SQL query
```

*Sam Lau and Sean Kross contributed equally to this work as co-first authors.

DataEd '23, June 23, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0207-5/23/06.
https://doi.org/10.1145/3596673.3596972

We note that SQL introduces additional challenges: its syntax is the *reverse* (but not quite) of the execution order, and an optimizer opaquely translates the SQL statement into a tree of physical operations. Understanding the conceptual and physical evaluation of a SQL query is a notorious stumbling block for many students.

Instructors like Alice must now resort to hand-drawing ad-hoc diagrams or making presentation slides to illustrate how these statements work, which can be tedious and time-consuming.

2) It is hard to understand how data science tools differ. Data science courses introduce multiple programming languages and tools (e.g., both pandas and SQL) as each is well-suited for different use cases. Students may expect that a pandas and SQL statement should be semantically equivalent, but be surprised that they are not. How can we allow them to see these subtle semantic differences?

In this paper we show that a language-independent step-by-step visual representation of data transforms can address these pedagogical challenges. To do so, we created a JavaScript-based table visualization library that illustrates the row-, column-, and cell-wise relationships between an operation's input and output tables. On top of this library we built a trio of freely-available web tools – Pandas Tutor for Python [14], Tidy Data Tutor for R [13], and SQL Tutor [28] – that automatically produce diagrams of how Python/R/SQL code transforms data tables step-by-step from input to output.

As a concrete example, Figure 2 shows one of our tools, Pandas Tutor, running the code from Figure 1. The user visits pandastutor.com and writes their Python code. Then Pandas Tutor automatically produces a diagram for each of the four transformation steps: a) the filter shows the correspondence between input rows that were retained by the filter predicate, and draws a box around the attributes used in the predicate; b) sort renders arrows to map input rows to their new positions in the output and draws a box around the sorting attribute; c) groupby draws a box around the grouping attribute and color-encodes rows in each group; d) aggregation shows how rows in each group map to individual output statistics.

Compared to Figure 1, the step-by-step diagram created by Pandas Tutor in Figure 2 makes it much easier for an instructor to explain to students what is going on behind the scenes to transform the input table to the final output table. Our other two tools – Tidy Data Tutor for R and SQL Tutor – work the same way. *To our knowledge, these tools are the first to render step-by-step diagrams of data table transformations that appear across multiple languages used in teaching introductory data courses.*

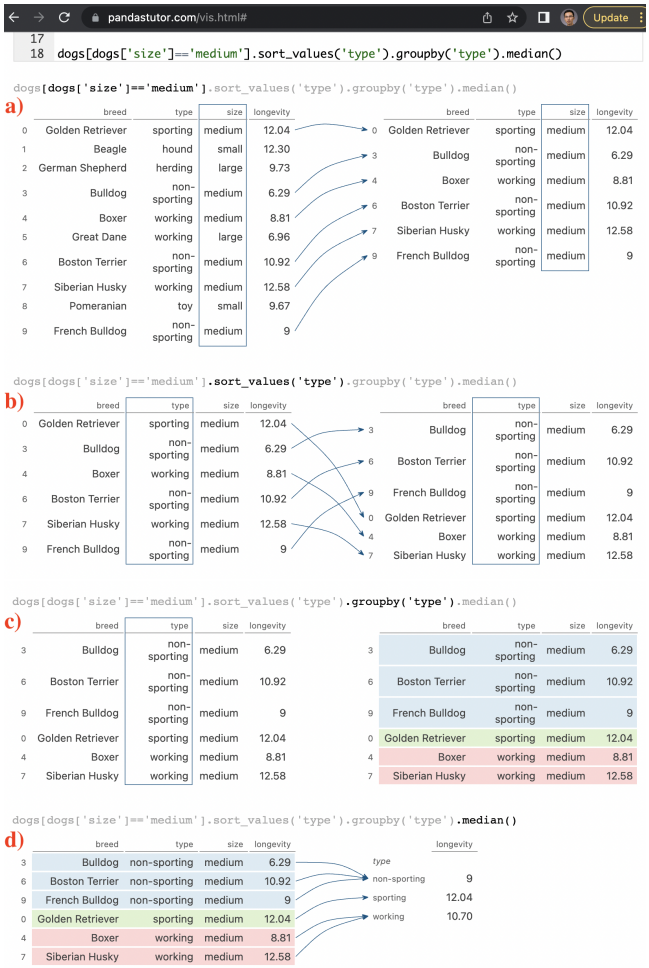


Figure 2: Pandas Tutor is a web application that automatically visualizes how Python pandas code transforms data tables step-by-step. This screenshot shows the example from Figure 1: a) filtering, b) sorting, c) grouping, d) aggregating. We also created analogous visualization tools for R and SQL.

2 RELATED WORK

The two closest related projects to ours – Data Tweening [12] for SQL and Datamations [21] for R – use animations to show how data tables get reshaped. We took a complementary design approach by rendering static diagrams that can be used in screenshots and presentation slides, and we also support a larger set of operators necessary to cover what is taught in introductory data courses.

Interactive data wrangling systems such as Wrangler [11], its proactive extension [7], and Unravel [23] show inline visual previews of table reshaping operations. These were designed to assist working data scientists; in contrast, our tools were made specifically for teaching, so they also show side-by-side before-and-after comparisons and fine-grained mappings to Python/R/SQL code.

Our work also extends the line of computing education research on program visualization tools [24]. These tools, such as Jeliot [17], UUhistle [25], and Python Tutor [6], visualize the step-by-step runtime state of code written for introductory programming courses. Our tools are inspired by Python Tutor and extend its reach to introductory data courses. This required us to design a different set of visualizations focused on annotating table transformations instead of on variables, objects, pointers, stack frames, and heaps.

Lastly, tools such as QueryVis [15], SQLVis [16], and SQL EXPLAIN [18] can help users to understand SQL query plans. These tools show the query plan and the *schemas* of affected data tables; but they do not show the concrete data that is being transformed, since their goal is to emphasize a higher level of abstraction. We build upon those ideas by showing both the query plan and the actual data tables on-demand when the user clicks on nodes in the query plan tree. Our intuition is that showing actual data can help instructors explain SQL execution more concretely to students.

3 SYSTEM DESIGN AND IMPLEMENTATION

Pandas Tutor, Tidy Data Tutor, and SQL Tutor are web-based tools that take the user's code and data as inputs (data can be passed in via a .csv file), runs the code on that data, and produces a set of before-and-after table transformation diagrams (see Figure 2).

Each tool has a language-specific backend that takes Python, R, or SQL code and adds precise run-time provenance/lineage tracking to it. For instance, Pandas Tutor uses LibCST [9] to rewrite and instrument calls to Python pandas filtering functions like Figure 2a to track the column(s) being filtered on and the rows that were selected. This instrumentation approach means that we must manually[1] add support for each function to track. While doing so may be impractical for supporting, say, the entire pandas or tidyverse libraries with hundreds of API functions, in practice we found that supporting around a dozen functions for each language was sufficient for teaching basic concepts in data science courses.

SQL Tutor's backend uses a Python-based educational DBMS called Databass [27] that is instrumented to track record-level lineage on a per-operator basis based on techniques from Smoke [20]. We run Databass in a web browser using Pyodide [1] to translate to WASM (WebAssembly). It supports SPJA queries, including nested subqueries. In practice, any DBMS that can export its query plan and per-operator lineage info can be used as SQL Tutor's backend.

Each tool's backend runs the user's instrumented code and records the before and after states of the table that is being transformed in each step along with tracked provenance about affected rows/columns/cells. That is why a single line of code like Figure 2 can produce four diagrams, since it contains four transformation steps in a pipeline (filtering, sorting, grouping, aggregating). This information then gets sent to the web-based frontend, which uses our core visualization library (see next sections) to display it on-screen.

3.1 Design of Core Table Visualization Library

All of our tools use a common core table visualization library that we wrote in JavaScript. Our main design principles were:

- **Show input-output correspondences** – Our library renders tables along with annotations such as bounding boxes, color highlights, and arrows between rows, columns, and cells (possibly across multiple tables). These annotations are critical for teaching how input and output data correspond, as shown by Figure 2. To avoid visual overload, users can mouse hover and click to selectively show/hide annotations.
- **Screenshot-friendly** – We wanted our visualizations to look polished enough so that users can take high-quality screenshots that they can put in presentation slides or lecture notes. We drew aesthetic inspiration from good hand-drawn diagrams that we saw online and from studying how expert instructors created their slides. Unlike related work such as Data Tweening [12] and Datamations [21] we choose not to use animations and instead render only static diagrams.
- **Compact** – What happens when users pass in tables that are too large to fit on the screen? In that case, our library uses heuristics to show the most relevant 12 rows by 8 columns – prioritizing those that are being operated on, truncating long strings, and sampling a few rows from each group (when grouping is used). Users can un-hide cells and long strings by dragging on hidden portions to reveal more data as needed.
- **Embeddable and shareable** – A JavaScript library makes it easy to embed visualizations on any webpage and to share as URLs. It also allowed us to embed Pandas Tutor into Jupyter Notebooks for Python and Tidy Data Tutor into RStudio for R since those data science IDEs are also built in JavaScript.

[1]We did consider automatically inferring provenance at the Python/R interpreter level. But false positives may produce inaccurate diagrams, which we did not want to risk.

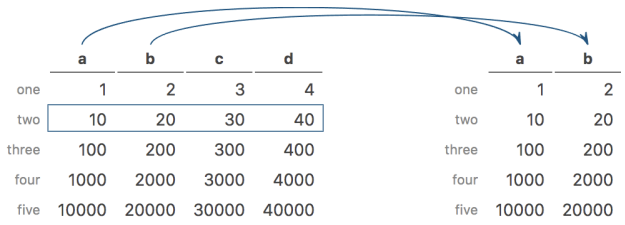
3.2 Supported Data Transformation Operators

On top of this core visualization library we implemented a set of interactive diagrams to teach common data transformations. In our experience, these have been sufficient to express the range of operators taught in introductory data science and databases courses. For each of these diagram types, we implemented API hooks into our Python/R/SQL backends so that when users run code in those languages, our tools call the visualization library to render the appropriate diagrams. Here are all of the supported diagram types:

Selecting and filtering: Our tools visualize how operators select individual rows/columns out of a table and optionally filter based on boolean conditions. In Python pandas, this is done via the bracket operator and other operators like `.get()`, `.loc[]`, and `.iloc[]`. In R tidyverse, it is done via the `select()`, `filter()`, and `mutate()` functions; and in SQL via the `SELECT` and `WHERE` clauses.

Figure 2a visualizes filtering rows based on a boolean condition (medium-sized dogs). Here is a more complex example of a line of pandas code that selects columns from a dataframe `df` filtered on the values of a specific row – `df.loc[:, df.loc['two'] <= 20]`

This kind of idiomatic pandas code with brackets, colons, `.loc[]`, and boolean conditions can be very hard for beginners to understand. Running it in Pandas Tutor clarifies what it does by showing how the `a` and `b` columns are selected because their values in the row labeled `two` (highlighted with a rectangle) are `<= 20`.



Sorting: Figure 2b shows how rows are sorted using the pandas `sort_values()` function. Similar diagrams are rendered for sorting using `arrange()` in R tidyverse and `ORDER BY` in SQL.

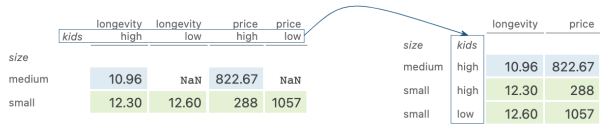
Grouping: Figure 2c shows the pandas `.groupby()` function creating three groups of rows, each with a unique color. Similar diagrams are rendered for `group_by()` in R tidyverse and `GROUP BY` in SQL. These diagrams use palettes from ColorBrewer [8] with 10 colors, which is usually enough for the small examples used in teaching. A warning appears if the user's code creates more than 10 groups.

Group-wise operations: After grouping, operations can be applied to all rows within each group. For instance, Figure 2 shows the pandas `.median()` function applied to numeric values within each group. R tidyverse has functions such as `summarize()`, `arrange()`,

and `filter()` that are group-aware and thus run within each group. And SQL has different `SELECT` parameters to accompany `GROUP BY`.

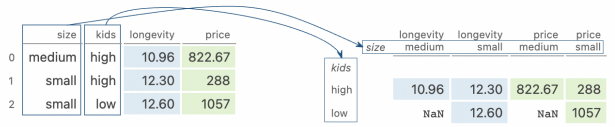
Reshaping: An important precursor to data analysis is *wrangling* [11] (or tidying [26]) raw datasets into a form that is amenable to analysis. This may involve reshaping tables to rearrange the orientations of their rows or columns. In our experience, table reshaping operations can be hard for students to understand because data suddenly moves around in non-intuitive ways.

Our tools use a combination of colors, highlights, and arrows to show how each reshaping operation works. For instance, here the pandas `.stack()` function 'rotates' cells around the `kids` label (called an 'index') and turns it from a column index to a row index:



Here is how the `.pivot()` function rotates a table from a 'long' to 'wide' format by turning `size` from a row to a column index:

```
dogs.pivot(index='kids', columns='size')
```



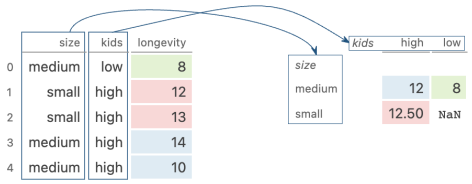
R tidyverse does reshaping via `pivot_longer()`, `pivot_wider()`.

What is especially challenging about teaching reshaping operators is that they can turn metadata in indices (e.g., row or column labels) into regular cell data and vice versa; also, multiple nested layers of indices (called hierarchical indexing [3] in pandas) may get created or destroyed in the process. Our tools visualize all of these intricate interactions between metadata and table cell data.

Pivot tables: The pandas `.pivot_table()` function aggregates data into a pivot table, an operation inspired by spreadsheets.

Here is how Pandas Tutor visualizes an example pivot table call where the `kids` row index pivots into being a column index and the values in the `longevity` column aggregate into a 2x2 cross-tab:

```
dogs.pivot_table(index='size', columns='kids', values='longevity')
```



Similar to reshapings, pivot tables can be hard to understand since passing in different values can result in very different outputs.

Joining two tables: All the above operators transform a single table, but our tools also show joins between two tables. In pandas this occurs via the `.join()` and `.merge()` functions, in R tidyverse via the `inner_join()`, `left_join()`, `right_join()`, and `full_join()` functions, and in SQL via variants of the `JOIN` clause.

Here Pandas Tutor visualizes a call to `.merge()` to left-join two tables via the `likes` and `breed` columns. The two input tables appear side-by-side while the output table appears below them. The columns to join on are surrounded with rectangles, and each

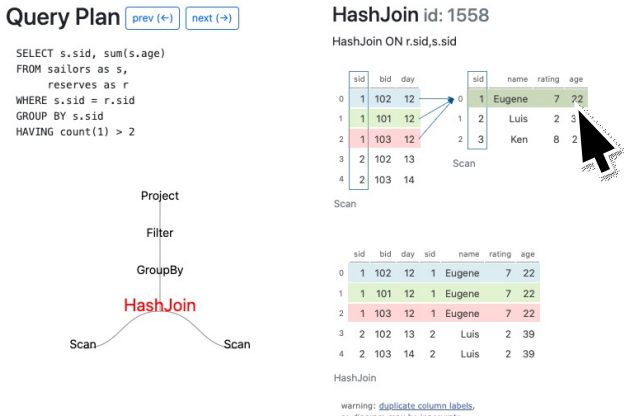
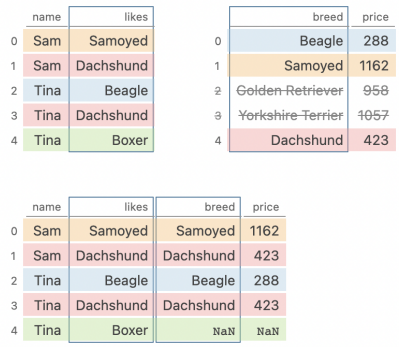


Figure 3: SQL Tutor visualizes a SQL statement's query plan (left) and lets the user step through its execution and interactively examine each operator's input and output tables along with their row, column, and cell-level dependencies (right).

row's color matches the row in the other table that it is being joined with:

```
people.merge(dogs, left_on='likes', right_on='breed', how='left')
```



	name	likes	breed	price
0	Sam	Samoyed	Samoyed	1162
1	Sam	Dachshund	Dachshund	423
2	Tina	Beagle	Beagle	288
3	Tina	Dachshund	Dachshund	423
4	Tina	Boxer	NaN	NaN

Since this is a left join, all rows in the left table make it to the output (bottom), but the Golden Retriever and Yorkshire Terrier rows in the right table do not make it since they have no matches in the left table. The user can tweak the code to change the columns to join on or to switch to a right join, inner join, or full outer join; then the visualization will update to illustrate the differences.

3.3 Visualizing SQL Query Plans

Python pandas and R tidyverse code statements execute in a linear sequence (i.e., a pipeline), so the corresponding visualizations can also be linear (e.g., Figure 2). However, SQL engines compile a single statement into a *tree* of physical operations, so this linear visualization is no longer sufficient. Thus, one challenge of learning SQL is that its declarative nature obscures the correspondence between the order of clauses in its syntax and the operators in the actual execution. In addition, the optimizer chooses from a large space of physical plans. To address this challenge, SQL Tutor [28] visualizes the step-by-step execution of a physical query plan.

Figure 3 shows its user interface. In contrast to Pandas or R statements, which form a linear sequence of operations, a physical query plan for a SQL statement forms a hierarchical structure. Thus,

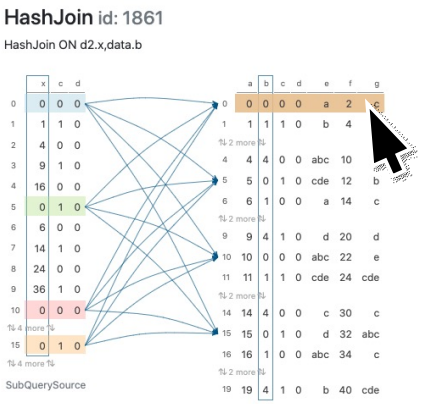


Figure 4: SQL Tutor visualizing a many-to-many join when the user hovers over a record in the right relation.

SQL Tutor explicitly shows the query plan tree on the left. To zoom into each step, users can click on an operator or step through via a depth-first tree traversal by clicking the previous/next buttons.

The right side of Figure 3 lists the operator name, internal id, and the operator's configuration (e.g., `HashJoin ON r.sid, s.sid`). The diagrams are nearly identical to those for pandas and R since all tools use the same visualization library. For instance, this example joins the sailors (left) and reserves (right) relations on `sid`. The user is hovering over the first 'reserves' record, which filters the visualization annotations to the sailors that join with this record. The arrows show the sailor rows that join with `Eugene`, and use the same colors as their corresponding output records in the bottom table. Vertical bounding boxes denote the join keys in both relations.

For many-to-many joins, all matching records on both sides are highlighted. For instance, Figure 4 shows that the user has highlighted the first record on the right relation; the tool then colors all join candidates in the left relation *and* draws arrows from those candidates to their additional matches in the right relation.

4 DEPLOYMENT AND PRELIMINARY IMPACT

We deployed `pandastutor.com` and `tidydatatutor.com` publicly in December 2021 and spread the word to fellow instructors via our professional and social networks. As a preliminary indicator of impact, Google Analytics shows that around 45,000 unique users visited `pandastutor.com` and 16,000 visited `tidydatatutor.com` between Dec 2021 and the end of Apr 2023 (17 months). These visitors came from over 160 countries spanning most of the world (see Figure 5). SQL Tutor is also available on the web [28], but we have not publicized it as widely since it is still under development.

Although these website visitor numbers from around the world are personally exciting to us, we acknowledge that they are not a substitute for conducting a formal evaluation to study what instructors do with these tools and whether it helps their students to learn better. As some early steps here, we collected the following anecdotes from our personal experiences teaching with these tools:

Co-lead-author Sam Lau used Pandas Tutor extensively while teaching DSC 10: Principles of Data Science at UC San Diego during Summer 2022. He observed that integrating Pandas Tutor within Jupyter Notebooks was essential for student adoption since the

Figure 5: Google Analytics data showing the approximate number of people per country who visited the Pandas Tutor and Tidy Data Tutor websites from Dec 2021 to April 2023.

course materials were all presented within Jupyter. He made custom stylistic adjustments such as font sizes and color contrast to make the visualizations more legible when presenting on a projector or via remote screen-share. Student feedback was positive, and he observed students using the tool during class to visualize more complex functions like groupby and merge. Other instructors of DSC 10 and also DSC 80: Practice and Application of Data Science at UC San Diego are now starting to use Pandas Tutor in class.

One limitation he encountered was the fact that Pandas Tutor cannot handle datasets larger than a few megabytes. While this is not a problem for demonstrating small in-class examples, homework assignments often involve larger datasets (e.g., tens to hundreds of MB) so students could not use Pandas Tutor to debug their homework code. Optimizing performance on larger datasets is one avenue for future work. (Tidy Data Tutor has the same limitation.)

More broadly, the Pandas Tutor and Tidy Data Tutor websites include a sign-up form to join a private instructor mailing list to receive updates on tool development. So far, over 90 data science instructors have joined the list and some wrote a note on their form about their interest in trying these tools in their classes. We hope to follow up with them to see whether they have used it.

Co-author Eugene Wu released an early version of SQL Tutor in his section of Columbia University's Introduction to Databases course in Fall 2022. Students in the course used it to visualize over 300 queries, with spikes that coincided with exams. One limitation of the current implementation is its reliance on a custom and incomplete SQL parser. Our plan is to replace this with an instrumented version of DuckDB [22] that captures the lineage needed to visualize the query execution. DuckDB is increasingly used in industry and data courses due to its easy-to-install nature, ability to run in the browser thanks to WASM compilation, and fairly complete feature set. Once we make this transition, SQL Tutor will be released and disseminated more widely.

5 CONCLUSION

Data science instructors find it challenging to explain to beginners how exactly Python/R/SQL code transforms tabular data. To help overcome this challenge, we created a set of freely-available web-based tools that visualize data table transformations step-by-step in three popular languages: Pandas Tutor for Python [14], Tidy Data Tutor for R [13], and SQL Tutor [28]. We are now working on integrating these tools more deeply into data science courses.

ACKNOWLEDGMENTS

Thanks to Robert Ward for the initial web prototype of SQL Tutor and to Rachel Lim for reading a draft of this paper. Thanks to UC San Diego for Philip Guo's sabbatical in 2021–2022, which led to the inception of this systems-building project. This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900; NSF grants 1845638, 1740305, 2008295, 2106197, 2103794; and a grant from the Alfred P. Sloan Foundation.

REFERENCES

[1] 2019. Pyodide is a Python distribution for the browser and Node.js based on WebAssembly. https://pyodide.org/. Accessed: 2023-02-20.

[2] 2022. 11 Types of Data Science Jobs (With Responsibilities). https://www.indeed.com/career-advice/finding-a-job/types-of-data-science-jobs. Accessed: 2023-02-20.

[3] 2023. pandas - MultiIndex / advanced indexing. https://pandas.pydata.org/docs/user_guide/advanced.html. Accessed: 2023-02-20.

[4] 2023. pandas - Python Data Analysis Library. https://pandas.pydata.org/. Accessed: 2023-02-20.

[5] 2023. Tidyverse: R packages for data science. https://www.tidyverse.org/. Accessed: 2023-02-20.

[6] Philip Guo. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 1235–1251. https://doi.org/10.1145/3472749.3474819

[7] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 65–74. https://doi.org/10.1145/2047196.2047205

[8] Mark Harrower and Cynthia A Brewer. 2003. ColorBrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal* 40, 1 (2003), 27–37.

[9] Meta Platforms Inc. 2019. LibCST - A Concrete Syntax Tree (CST) parser and serializer library for Python. https://github.com/Instagram/LibCST. Accessed: 2023-02-20.

[10] Alexander C. Kafka. 2018. With Student Interest Soaring, Berkeley Creates New Data-Sciences Division. *The Chronicle of Higher Education* (Nov 2018).

[11] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 3363–3372. https://doi.org/10.1145/1978942.1979444

[12] Meraj Khan, Larry Xu, Arnab Nandi, and Joseph M. Hellerstein. 2017. Data Tweening: Incremental Visualization of Data Transforms. *Proceedings of the VLDB Endowment* 10, 6 (2017), 661–672.

[13] Sean Kross and Philip J. Guo. 2021. Tidy Data Tutor - visualize R tidyverse data pipelines. https://tidydatatutor.com/. Accessed: 2023-02-20.

[14] Sam Lau and Philip J. Guo. 2021. Pandas Tutor - visualize Python pandas code. https://pandastutor.com/. Accessed: 2023-02-20.

[15] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, H. V. Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based Diagrams Help Users Understand Complicated SQL Queries Faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2303–2318.

[16] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual query representations for supporting SQL learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–9.

[17] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. 2004. Visualizing Programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Gallipoli, Italy) *(AVI '04)*. Association for Computing Machinery, New York, NY, USA, 373–376. https://doi.org/10.1145/989863.989928

[18] MySQL. 2023. EXPLAIN Statement. https://dev.mysql.com/doc/refman/8.0/en/explain.html. Accessed: 2023-02-20.

[19] Devin Petersohn, Stephen Macke, Doris Xin, William Ma, Doris Lee, Xiangxi Mo, Joseph E. Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya Parameswaran. 2020. Towards Scalable Dataframe Systems. *Proc. VLDB Endow.* 13, 12 (jul 2020), 2033–2046. https://doi.org/10.14778/3407790.3407807

[20] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-Grained Lineage at Interactive Speed. *Proc. VLDB Endow.* 11, 6 (feb 2018), 719–732. https://doi.org/10.14778/3199517.3199522

[21] Xiaoying Pu, Sean Kross, Jake M. Hofman, and Daniel G. Goldstein. 2021. Datamations: Animated Explanations of Data Analysis Pipelines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[22] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1981–1984. https://doi.org/10.1145/3299869.3320212

[23] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Unravel: A Fluent Code Explorer for Data Wrangling. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 198–207. https://doi.org/10.1145/3472749.3474744

[24] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans. Comput. Educ.* 13, 4, Article 15 (nov 2013), 64 pages. https://doi.org/10.1145/2490822

[25] Juha Sorva and Teemu Sirkiä. 2010. UUhistle: A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10)*. Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/1930464.1930471

[26] Hadley Wickham. 2014. Tidy data. *Journal of Statistical Software* 59, 10 (2014), 1–23.

[27] Eugene Wu. 2020. databass is a query compilation engine built for Columbia's database courses. https://github.com/w6113/databass-public. Accessed: 2023-02-20.

[28] Eugene Wu. 2022. SQLTutor Visualizes Query Execution. https://cudbg.github.io/sqltutor/. Accessed: 2023-02-20.