Detangler: Helping Data Scientists Explore, Understand, and Debug Data Wrangling Pipelines

Nischal Shrestha
North Carolina State University
Raleigh, North Carolina, USA
nshrest@ncsu.edu

Bhavya Chopra
Microsoft
India
t-bhchopra@microsoft.com

Austin Z. Henley
Microsoft
Redmond, Washington, USA
austinhenley@microsoft.com

Chris Parnin
Microsoft
Raleigh, North Carolina, USA
chrisparnin@microsoft.com

***Abstract*—Data scientists spend significant time on data wrangling—a process involving data cleaning, shaping, and pre-processing. Data wrangling requires meticulous exploration and backtracking to assess data quality by applying and validating numerous data transformation chains, making it a tedious and error-prone process. In this paper, we present Detangler, an interactive tool within the RStudio IDE that helps data scientists identify and debug data quality issues and wrangling code. The design of Detangler is informed via formative interviews, and it presents data scientists with (i) insights into potential data quality issues, and (ii) always-on visual summaries of the effects of individual data transformations, enabling interactive exploration of data and wrangling code. Through a laboratory study with 18 data scientists, triangulated with telemetry data, we find that Detangler improves exploration and debugging of data smells and data wrangling code. We discuss design implications for future tools for data science programming.**

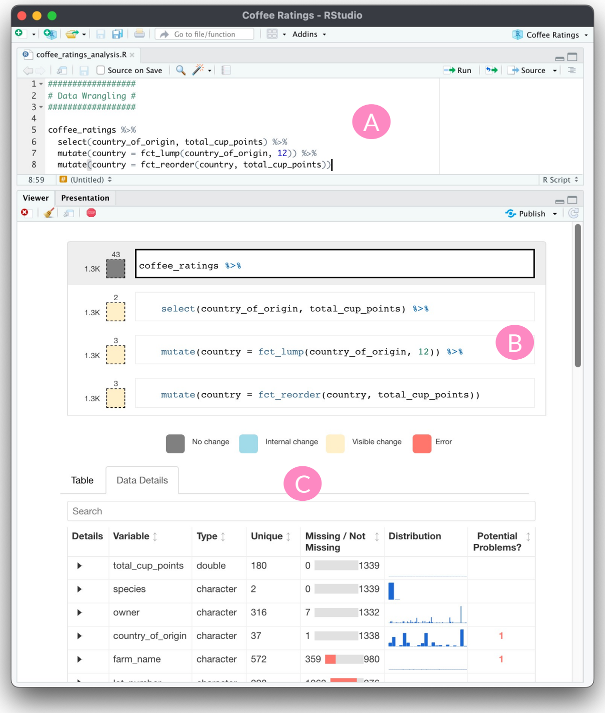
***Index Terms*—data science, data wrangling, programming**

I. Introduction

The general workflow of a data science project involves data discovery, collection, ingestion, cleaning and pre-processing, exploratory data analysis, and data modelling [1]. The advent of big data and data-driven systems make data wrangling an increasingly necessary and demanding task, as real-world data is riddled with varying data quality issues (missing values, inconsistent formatting, duplicate values, and so on). Data wrangling is a time-consuming, tedious, and error-prone process, requiring validation, debugging, and continuous iterations over data transformation chains [2, 3, 4, 5]. For data modelling tasks, data scientists must also iterate on their feature selection, architecture, and hyperparameters [6].

Data scientists commonly use the fluent interface programming pattern [7, 8], which involves composing multiple operations into a chain. Each operator in the chain accepts data resulting from the previous operation, performs a transformation or computation on this data, and passes the result to the next stage of the chain. This pattern is supported using the `tidyverse`[1] package's pipe (%>%) operator in R. However, isolating problems in a broken chain is cumbersome and riddled with errors — requiring manual inspection of intermediate results. This is exacerbated when data already comes with issues such as missing values and outliers, that affect data analysis downstream. Another key characteristic of data wrangling code

[1]https://www.tidyverse.org/



```r
##################
# Data Wrangling #
##################

coffee_ratings %>%
  select(country_of_origin, total_cup_points) %>%
  mutate(country = fct_lump(country_of_origin, 12)) %>%
  mutate(country = fct_reorder(country, total_cup_points))
```

🟫 No change 🟦 Internal change 🟨 Visible change 🟥 Error

Details	Variable	Type	Unique	Missing / Not Missing	Distribution	Potential Problems?
▶	total_cup_points	double	180	0 1339		
▶	species	character	2	0 1339		
▶	owner	character	316	7 1332		
▶	country_of_origin	character	37	1 1338		1
▶	farm_name	character	572	359 980		1

Fig. 1: Detangler can be invoked from the code editor (A) to validate transformations and detect data smells. Users can explore intermediate results using the code overlay (B). Data Details and Table views display potential data quality issues and always-on visuals for the selected transformation step (C).

is that it involves "exploratory" programming [9], producing messy code that can introduce data-related issues that go unnoticed until an analysis is performed. To deal with these issues in data wrangling pipelines, data scientists are forced to perform (i) line-by-line code isolation to validate individual transformation steps, (ii) manual inspection of datasets, and (iii) data quality checks by writing additional code.

Our preliminary research informs us that data scientists heavily rely on manual data inspections to identify data quality issues. Few data scientists leverage data exploration libraries, such as `skimr` for R and `pandas-profiling` for Python, that provide descriptive statistics and summaries based on the data. However, data analysts may not always be aware of potential data smells and quality issues, such as the presence of miscoded

`NA`s (such as '-' in place of NaN), typing-errors in categorical columns, outliers, and unexpected values, which lead to errors downstream. Further, to use such tools, data scientists have to go out of their way to understand and learn framework specific end-points to write relevant data quality checks.

Towards addressing these shortcomings, we designed Detangler—an in-situ tool for the RStudio IDE for data scientists to explore, understand, and debug data wrangling code and data smells. Detangler presents 'Data Details' and 'Table' views (as seen in Figure 1) that enable data scientists to step through their wrangling pipelines and discover important characteristics about their data through (i) alerts for potential data quality issues, (ii) always-on visualizations, and (iii) visual representations of data modifications. With these features, Detangler aims to reduce manual efforts involved in discovering data smells and validating code behaviour. Through a usability evaluation of these design features, we find that Detangler supports data scientists in both, exploration, as well as debugging of data quality issues and mistakes in the code. The contributions of this paper are the following:

1) Findings from a formative study with 8 data scientists—surfacing pain-points in data wrangling workflows, and design considerations for data science tools.
2) The design of an in-situ interactive tool, called Detangler, that provides data scientists the agency to perform data quality checks and structured explorations of data wrangling code at each transformation step.
3) Findings from a laboratory study with 18 data scientists, demonstrating that Detangler reduces data scientists' data wrangling efforts by reporting potential data smells and validating the effects of data transformation pipelines.

II. Related Work

A. Tools to manage wrangling code and output versions

Researchers have developed tools that help data scientists write and modify code. Kery and Myers [9] found that data science programming is characterized by exploratory programming, which can lead to disorganized and ephemeral code. Tools like Gather [10] help find, clean, recover, and compare versions of code in cluttered, inconsistent computational notebooks like Jupyter. To explore alternative code in notebooks, Fork It [11] uses a technique to fork a notebook and directly navigate through decision points in a single notebook. To help data scientists automate writing data wrangling code, Wrex [12] uses programming-by-example. Similarly, Mage [13] helps users generate code based on the modifications of their dataframes. We build on the promising design of Unravel, a tool that facilitates the understanding of fluent code through structural edits [14]. In addition to performing exploratory tasks with the fluent programming pattern, Detangler aims to help data scientists identify potential data quality issues and validate transformation chains.

B. Exploring and understanding data science programs

Prior work has explored some tools to help data scientists understand code. For example, TweakIt is a system designed to help end-user programmers collect, understand, and tweak Python code in a spreadsheet [15]. There are also tools that help data scientists visualize how common data wrangling operations work. The Datamations tool animates dataframe wrangling and visualization pipelines in R. Datamations automatically processes fluent code in R using `tidyverse` packages and provides a paired explanation and visualization of each step in the chain [16]. Detangler is designed as an interactive tool used within RStudio IDE to allow opportunistic learning and debugging of data wrangling code with the use of always-on visuals [17, 18]. There are also web-based tools like Tidy Data Tutor that visualizes functions in R to help data scientists understand how those functions transform data [19]. Another tool, DS.js, provides end-users with a data science environment within web-pages, enabling them to interactively explore structured data [20]. `pipediff` [21] is an RStudio plugin that highlights the differences between two adjacent steps in a data transformation pipeline within the IDE. However, fewer tools are available for data inspection and debugging.

C. Debugging data wrangling code and data smells

More recently, there has been some focus on data quality issues and tooling to fix them. Diff In The Loop (DITL), emphasizes the idea of displaying data and distribution differences for versions of data wrangling code [22] during exploratory analysis work. Our work is most closely aligned with DITL with regards to problem motivation and some aspects of the design. While DITL uses the idea of code snapshot differences, we focus on transformation differences in a data wrangling pipeline for one version of the code at a time. The DITL prototype also shows various descriptive statistics and visualizations like histograms and their differences between these code snapshots, which can be useful for debugging transformations. Another web-based tool, Rill Developer [23], automates data quality checks for SQL by stripping away the need to perform the manual checks and visualizing issues like missing values.

III. Formative Interviews and Design Goals

To better understand the common pain points of exploring data wrangling code and the types of data quality issues that are commonly dealt with, we conducted semi-structured interviews with 13 data scientists. We recruited 8 data scientists (F1–F8) who frequently use the RStudio IDE to wrangle data in R. The discussions focused on how they perform data wrangling, how data wrangling fits within their IDE workflow, what tools they use (or have used) for checking data quality issues, and what difficulties they face as they wrangle data. These insights guided the design goals for Detangler.

1) Manual inspection of data for quality issues: All of the data scientists made heavy use of features within RStudio that help them explore data. F4, F5, and F8 talked about how they typically start by previewing the first or last few rows of the data and the column or variable types. However, this is typically insufficient since console outputs in IDEs like RStudio *"may not reveal potential issues with data like missing values"* (F4). Similarly, F1, F2, and F3 talked about how they also have difficulty *"catching subtle issues"* (F1) like specific values that one has to normally write code to filter out. When exploring data for the first time, all data scientists described common checks that they perform manually by writing code such as checking for variable types (e.g. string versus a number), missing values (`NA`s), and miscoded [24] `NA` values (e.g. -99, "-", etc.), outliers, unexpected values, distribution of variables, and descriptive statistics (e.g. ranges).

Several participants mentioned using built-in functions in R such as `is.na` or packages like `skimr` which provides a static printout of the column types, and their descriptive statistics. F3, F4, and F5 described using `skimr` to summarize checks and descriptive statistics into a text output, but this still didn't allow *"interactive introspection into each of these characteristics"* (F3). F1, F4, F5, and F6 found histograms of variable distributions to be a quick, easy and reliable way to understand the characteristics of the dataset as a whole. These insights motivate our first design consideration:

> **D1:** Data wrangling tools should enable insight finding and detection of data smells, outliers, and unexpected values without requiring manual inspection of the dataset.

2) Validating changes made to data for individual transformations: Data wrangling code typically consists of moderately long transformation pipelines. Data scientists reported trouble in isolating individual transformation steps to understand and validate changes that they have made to the data. For exploration of data wrangling code and output, data scientists expressed that current tools make it difficult to navigate the many steps in a transformation pipeline, and understanding how the code and output corresponds with each other. Data scientists expressed that a significant portion of their time is spent on isolating and validating each transformation on the data. This leads us to the following design considerations:

> **D2:** Data wrangling tools should make it easy to preview and validate the output of individual transformations to avoid line-by-line code isolation.

> **D3:** Data wrangling tools should help data scientists find mistakes in their code and include automated descriptive statistics and checks for data transformations.

3) Frequent context switches within the IDE hinder data scientists' workflow: The process of debugging wrangling code is made difficult due to the absence of interactive ways to explore datatables. A few data scientists (F2, F4, F8) expressed that while they do make use of interactive tables in RStudio using the `View` function, it can be hard to manage multiple ad-hoc tables in the IDE. This leads to multiple views and tools used for exploratory checks that can take a data scientist out of context from their data wrangling code. This insight motivates the following design goal:

> **D4:** Data wrangling tools must make it easy to navigate to interactive data views and transformation step summaries in a consolidated view to reduce context switching.

IV. Detangler: Tool Design and Implementation

In this section, we present the implementation details and system design for Detangler with two innovative features to support data scientists wrangling workflow: the **'Table'** and **'Data Details'** views. Detangler's design is focused on facilitating wrangling transformations—aiding exploration and debugging of data smells and fluent code. Downstream analysis tasks such as visualizations and training are out of scope for Detangler. Detangler is a web-based tool that can be run within the RStudio IDE to explore, understand, and debug data wrangling code in R. We decided to prototype this tool for R as it is widely adopted and used for data science programming tasks, and is a popular language in general[2]. Detangler works with any transformation chains applied to dataframes (not specific to `tidyverse`, though it is popular). We discuss the design decisions to support design goals for Detangler, informed by the formative interviews in Section III.

1) Invoking Detangler to explore data wrangling code: Fluent interfaces are popularly leveraged by data analysts to apply chained transformations while maintaining the legibility of their code. To support data wrangling code exploration in this mode, Detangler can be invoked by piping code to the `detangle()` API as the final step in the transformation chain:

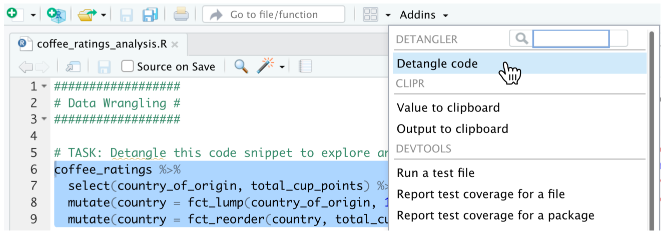
```
coffee_ratings %>%
select(country_of_origin, total_cup_points) %>%
mutate(country=fct_lump(country_of_origin, 12)) %>%
detangle()
```

Alternatively, Detangler can be invoked by highlighting the pipeline one wishes to *detangle*, and selecting 'Detangle' from the Addins drop-down menu, as seen in Figure 2. Detangler then opens up in the RStudio Viewer Pane[3], presenting users with an interactive **'Code Overlay'** for exploring pipeline stages. To provide high-level insights into intermediate results in the transformation pipeline, the overlay displays dataframe dimensions corresponding to each step, and visual cues for their execution. Detangler uses a *color schema* to indicate the nature of a change, as seen in Figure 1. Grey indicates 'No change', blue indicates an 'Internal change', yellow indicates a 'Visible change' to the dataframe, and red indicates an 'Error' in executing the transformation pipeline stage. To further support D4, each of the views present information using in-line always-on visualizations within RStudio's Viewer Pane.

We support interactive exploration of fluent data wrangling code by updating the Table and Data Details views with intermediate data information corresponding to the currently focused line of code in the overlay. Data scientists can thus select the data transformation steps they wish to analyse, avoiding the overhead of line-by-line code isolation for analysis (supporting D2).

2) Data Details: To support the design considerations (D1–D4), we implemented an the 'Data Details' view (Figure 3). In our formative interviews, several informants expressed the need for quick summary statistics for dataframe variables. Along with automating the summary of variable characteristics, data scientists also desired reducing the manual writing effort to perform data quality checks (detailed in Section III). The Data Details view supports these requirements, and displays

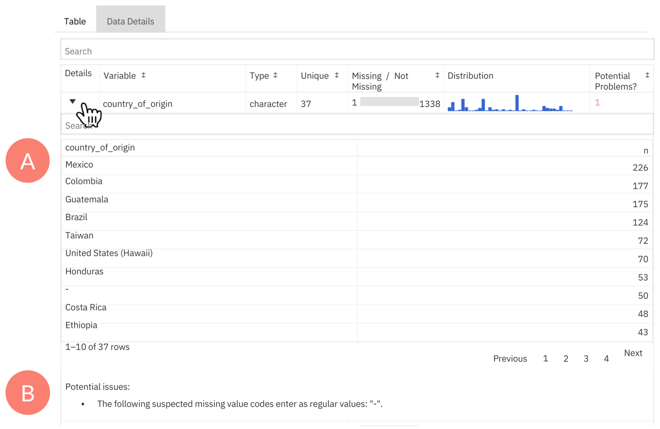
[2]TIOBE Index rankings for R: https://www.tiobe.com/tiobe-index/r/
[3]https://rstudio.github.io/rstudio-extensions/rstudio_viewer.html



```
##################
# Data Wrangling #
##################

# TASK: Detangle this code snippet to explore ar
coffee_ratings %>%
  select(country_of_origin, total_cup_points) %>
  mutate(country = fct_lump(country_of_origin, 1
  mutate(country = fct_reorder(country, total_cu
```

Fig. 2: To invoke Detangler, users can highlight the pipeline they wish to *detangle* and select 'Detangle' from the RStudio Addins drop-down menu.



Details	Variable	Type	Unique	Missing / Not Missing	Distribution	Potential Problems?
▼	country_of_origin	character	37	1 1338		1

country_of_origin	n
Mexico	226
Colombia	177
Guatemala	175
Brazil	124
Taiwan	72
United States (Hawaii)	70
Honduras	53
-	50
Costa Rica	48
Ethiopia	43

1–10 of 37 rows Previous 1 2 3 4 Next

Potential issues:

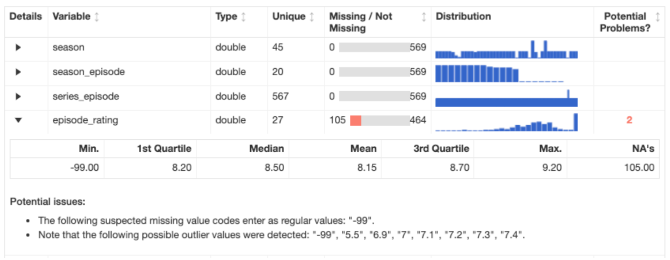
- The following suspected missing value codes enter as regular values: "-".

Fig. 3: A user can examine expanded details about a dataframe column, displaying type-specific statistics, such as count table (A), and potential issues (B) for the type such as miscoded NAs like "-" for categorical columns.

descriptive statistics and visuals for the intermediate data depending on the transformation step selected in the overlay. The view displays the (i) type, (ii) number of unique elements, (iii) number of missing elements, (iv) the distribution plot, and (v) alerts for potential problems for the selected column.

We were inspired by existing tools like `skimr` [25] that print descriptive statistics of variables in a dataframe, and use existing built-in R functions like `summary` for numeric variables (Figure 4), and functions like `count` from the `dplyr` package for counting values in categorical variables. Each column displays additional details describing type-specific (numerical versus categorical) statistics, and the potential issues data scientists may have to fix before modelling their data (Figure 3).

3) Table view: To further support the design considerations (D1–D4), we designed the Table view to supplement the Data Details view, by adding cues on modified columns, grouped dataframes, column types, and search by value. We added type information for each column of the table, like <`int`> for integer or <`chr`> for character, mimicking the console output for dataframes when using the `tidyverse` R packages. Participants of our formative interviews expressed how grouping data according to certain variables and performing aggregate computations—such as finding the average for each group—is a common pattern in data wrangling. To support this, the color change schema "Internal change" highlights



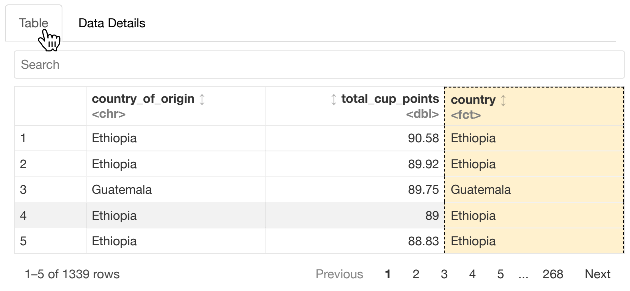
Details	Variable	Type	Unique	Missing / Not Missing	Distribution	Potential Problems?
▶	season	double	45	0 569		
▶	season_episode	double	20	0 569		
▶	series_episode	double	567	0 569		
▼	episode_rating	double	27	105 464		2

Min.	1st Quartile	Median	Mean	3rd Quartile	Max.	NA's
-99.00	8.20	8.50	8.15	8.70	9.20	105.00

Potential issues:

- The following suspected missing value codes enter as regular values: "-99".
- Note that the following possible outlier values were detected: "-99", "5.5", "6.9", "7", "7.1", "7.2", "7.3", "7.4".

Fig. 4: Visualizing numeric data with the Data Details view. Detangler provides descriptive statistics and potential issues for each column.



	country_of_origin <chr>	total_cup_points <dbl>	country <fct>
1	Ethiopia	90.58	Ethiopia
2	Ethiopia	89.92	Ethiopia
3	Guatemala	89.75	Guatemala
4	Ethiopia	89	Ethiopia
5	Ethiopia	88.83	Ethiopia

1–5 of 1339 rows Previous 1 2 3 4 5 ... 268 Next

Fig. 5: The Table view supplements the Data Details view, by adding cues for modified columns, grouped dataframes, column types, and search by value. Here, the `country` column is highlighted in yellow to indicate a visible change for the selected transformation step.

columns as blue when users are grouping variables, as opposed to "Visible change" where actual modifications are made to the dataframe. Detangler's Table view displays the dataframes whenever a `group_by` operation is performed.

Data scientists in our formative study found it easy to miss erroneous values that don't meet their expectations, such as placeholder values and miscoded NAs (Section III). To facilitate identification of such problematic values, we added search bars for the Table and Data Details views, enabling them to locate specific values (matched via pattern matching).

V. Laboratory Study

We conducted a usability study over video conference with 18 data scientists to understand how Detangler can assist them in identifying data smells, understanding and debugging data wrangling code for data analysis, and assessing if Detangler fits optimally in their data wrangling workflow.

A. Participants

We recruited participants through an online advertisement of the study on Twitter where data scientists are increasingly active [26]. To be eligible for the study, participants had to self-report their experience with data wrangling, R programming, and their field of work. Out of 123 sign-up survey responses, we recruited 18 data scientists (10 males, 8 females) via random sampling, who had varying levels of experience in data wrangling and programming in R. On a 5-point Likert scale, participants self-reported their experience in data wrangling (μ

= 3) and R (μ = 3.1), with a minimum of 1 and maximum of 5. The recruited participants belonged to varying fields, including software, psychology, and education. We used the average self-ratings of participants' data wrangling and R skills to bucket participants into either of the two categories: beginner data scientists (2–3.5) and experienced data scientists (above 3.5).

B. Protocol

The study was conducted remotely, over a video conferencing tool, where participants shared their screens for the session's duration. For the programming environment, we used RStudio Cloud IDE, a web-based version of the RStudio IDE. Being a browser-based IDE, participants were able to use Detangler on their computers within their own choice of browsers and configurations. Each session lasted 60 minutes, and consisted of a demo session and three debugging tasks.

1) Debugging Tasks: For the three debugging tasks, the participants were tasked with exploring analysis scripts written in `tidyverse` R using the `dplyr` and `tidyr` packages with the goal of fixing the data wrangling code to produce the desired visualization. The three datasets were based on #TidyTuesday datasets [26], which included data quality issues inherently, and we further modified them to include common data smells discussed in prior work [27], and common issues mentioned by participants (F1–F8) in our formative interviews. No participant indicated familiarity with the selected datasets.

Each task script was scaffolded with code to import the dataset, wrangle the data, and visualize it to explore relationships between variables. Participants were tasked with thinking aloud while exploring, understanding, and debugging the data wrangling code using Detangler. We did not prevent participants from writing their own code, in an attempt to investigate at which stage(s) participants would reach for Detangler during their data wrangling workflow. We designed the following 3 tasks including inherent issues with the data as well as varying mistakes in the data wrangling code:

(Task A) Coffee Ratings: Participants examined a script that visualizes total coffee ratings for countries around the world using a box plot. The provided data wrangling code involved filtering out missing values (`NAs`) or miscoded values like white space or 999. The missing values are not handled in data wrangling code which produces an incorrect plot containing box plots for total cup points by top 12 countries. The inclusion of miscoded `NAs` makes this process harder because they are no longer detected with explicit checks for `NAs`.

(Task B) Chopped: Participants examined a script visualizing the average episode ratings for all seasons of the Chopped TV Show. The provided data wrangling code included an incorrect order of summary statistics about particular groups of variables before grouping the data by those variables. Grouped calculations is common yet it can be confusing because it collapses large tables into a smaller ones by aggregating values according to groups. The data also contained a problematic value (-99) that fell outside of the range of expected values for a rating (0–100).

(Task C) Crop Yields: Participants examined a script that analyzed the global crop yields for major countries like USA, Brazil, China, and Russia. The data wrangling code reshaped the dataframe from a wide form (many columns) to a long form (many rows). However, the existing code contains two errors: incorrect column name being used with `pivot_longer` and incorrect data type—`year` column was converted from a `numeric` type to a `character` (string).

C. Data Collection and Analysis

All participant sessions were recorded and transcribed. We made field notes during the sessions to record observations about which features were used by participants, and in what ways, to facilitate exploration and understanding of the data wrangling tasks. We then performed inductive qualitative coding to analyze initial patterns and draw connections to derive axial themes that describe our participants' interactions with Detangler. We followed the guidelines set by Carlson [28] and performed a single-event member check with our results to overcome theoretical sensitivity. Following the completion of the usability session, we administered an exit survey to measure the usefulness of Detangler and its specific features.

To triangulate our findings, we also analyzed the log events for patterns that help explain our themes. We instrumented Detangler and RStudio to log all successful user code executions, Detangler invocations (by clicking lines in the code editor), counts of "focusing" on the views (via mouse hover event), and Detangler-specific feature usage.

VI. Results

In this section, we present themes and log analysis results from the user study, describing the behaviors observed, strategies used, and feedback from participants. The results of our user study suggest that Detangler addresses the design goals we formulated in Section III. Participants found that Detangler helped them explore data wrangling code as a consolidated view when they felt confused about transformations (D2, D4). Detangler enabled participants in further understanding, and more importantly, in catching issues with both—the data wrangling code and the data (D1, D3). Detangler made exploration of data wrangling code and output easier as *"it offers a lot of information in a very compact way."* (P14) Several features of Detangler helped participants interactively explore the code behavior and the dataframe transformations.

We observe that Detangler was used in bursts whenever participants were trying to understand code behavior or the resulting dataframe from the wrangling. Beginners and experienced participants did not differ much on their frequency of using Detangler or executing code in RStudio. Beginners *detangled* code 124 times and executed code 268 times in RStudio; while experienced participants *detangled* code 127 times and executed code 294 times.

A. User-Survey Results

Overall, data scientists' response to Detangler was positive. On a 5-point Likert scale (Table I), participants positively rated the usefulness of Detangler overall (μ = 4.5, median = 5). Participants rated all features positively: the Data Details view to catch potential data quality issues (μ = 4.7, median = 5) and understanding transformations (μ = 4.4, median = 5), and the always-on visualization showing dataframe shape (μ = 4.5, median = 5). Participants found code highlighting to invoke Detangler (μ = 4.2, median = 4) for viewing intermediate results helpful (μ = 4.6, median = 5).

B. Detangler provides quick validation of data characteristics

With a consolidated view presenting users with always-on visualizations, Detangler gives data scientists the agency to (i) form mental models of variable distributions, (ii) quickly and reliably validate the effect of individual transformations, and (iii) visually identify data smells and quality issues. Detangler successfully mitigates the need to perform manual inspections of data and saves the previously spent effort in writing code to perform data quality checks (supporting D1–D4).

1) Detangler's consolidated experience is inviting for data scientists: We observed a mix of beginners and experienced participants making use of RStudio features while examining data, such as the Environment Pane (P5, P7), the interactive table using the `View()` function (P5, P6, P9, P10, P18) that holds all variables such as the dataset variables, and the interactive console (P4, P5, P10). When participants were using these other views, they eventually decided to use Detangler when they felt the need to view both, the code and the output in the same window, supporting D4. Participants then made use of the always-on visualizations within Detangler (described in Section IV) to help them validate transformation pipelines.

2) Detangler facilitates isolation of transformations to validate assumptions about code behaviour: P13 favored the Detangler's Table view when checking dataframe outputs because of the ease of exploring them in the tool: *"I'm a very slow coder, I change things and come back and mess things up each time. [In Detangler], I can see that I'm messing up if my dataset has a strange shape and not go over to the console. I check the console output but it's not as easy to understand as what happened [on the console]."* Experienced participants like P5, P13, and P14 liked the fact that each line comes with its own output as well as Data Details, a clear difference from the traditional approach of writing a single expression pipeline and only seeing the final result: *"What's cool is that what happens below [on the table] depends where you are on the [code line] so it's for each line. Usually I only look at beginning data and final output."* (P5)

3) At-a-glance visualizations help data scientists form instincts: Participants used the Data Details view to get an overall sense of the columns and quickly catch issues. The unique number of elements and 'missingness' insights in the Data Details view helped participants quickly identify and validate issues corresponding to each transformation. P14 used the unique counts for the `season` column while trying to check how the counts dropped in Task B and was able to spot the missing values in episode ratings that dropped 2 seasons. Participants used the in-line histograms to spot extreme values.

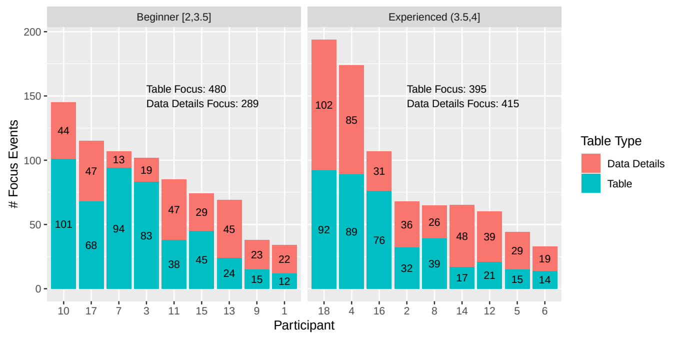


Fig. 6: Number of times participants focus on Table and Data Details views.

For example, P10, P12, and P13 looked at the histogram of the `total_cup_points` column in Task A and saw that it was heavily left skewed. They examined the expanded details for the column and noticed there was a 0 as a minimum value, and they were able to filter out that value. P14 was able to spot the incorrect ordering of the `group_by` after `summarize` by taking a peek at the histogram for the `season` column, and said, *"Yeah so that should definitely be more than 14 and should be more of a uniform distribution."*

C. Using Detangler to debug data wrangling mistakes and data quality issues

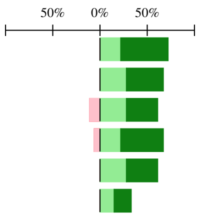
Detangler helped participants identify and debug data quality issues and code mistakes (supporting D1 and D3).

1) Data scientists organically switch between the views: Beginners made heavier use of the Table view (488), while still consulting the Data Details (300). Experienced participants equally split their time between both views, 436 in Table view and 422 in Data Details view (see Figure 6). Several beginners (P1, P2, P7, P9, P11) used the Table view and clicked through the 'Next'/'Previous' buttons to flip through the rows to validate assumptions for a particular column. For example, P11 flipped through some rows in the Table view when they were investigating the '-99' outlier in Task B and once that became laborious, they switched to the Data Details view which pointed out the value. Although hesitant at first, experienced participants like P4, P5, P8, P12 went directly to the Data Details view for every task to save on time after having success with it initially, instead of performing manual checks with the Table view.

2) Participants triangulate potential issues using multiple views: Participants used multiple views to understand potential issues, toggling between (i) transformation steps using the code overlay, (ii) the Data Details view for descriptive statistics and potential issues with attributes, and (iii) the Table view to look for values corresponding to those issues (e.g. by searching for specific values in the Table view). For example, P1 searched for the miscoded `NA` value of "-" in Task A for the `country` column in its expanded details within Data Details. Similarly, P2 made use of the search feature in Task C to validate whether the `crop` column contained values with an "_" in the name (`cocoa_beans`). P16 also made heavy use of the search feature

TABLE I: Post-Study Survey Responses

	% Agree	Likert Resp. Counts[1]					Distribution[2]
		SD	D	N	A	SA	
Data Details view helped identify potential issues with data.	94%	0	0	1	4	13	
Clicking lines to view intermediate data was useful.	94%	0	0	1	5	12	
Data Details view helped describe transformations.	89%	0	2	0	5	11	
The dataframe shape information helped validate changes.	89%	0	1	1	4	12	
Detangler was useful overall.	89%	0	0	2	5	11	
Invoking Detangler through code highlight and Add-in was useful.	83%	0	0	3	9	6	



[1] Likert responses: Strongly Disagree (SD), Disagree (D), Neutral (N), Agree (A), Strongly Agree (SA).
[2] Net stacked distribution removes the Neutral option and shows the skew between positive (more useful) and negative (less useful) responses.
🟥 Strongly Disagree, 🟥 Disagree, 🟩 Agree; 🟩 Strongly Agree.

to validate removing the "-" in the `country` column in Task A. When things still didn't make sense, participants explored other lines in the code overlay, their corresponding code, and the Table and Data Details views. For example, P14 saw how the unique counts for the `season` column in the last step was 43 instead of the original 45. Confused, they checked the line before the `filter` step and saw the missing values in episode ratings from the missing bar which dropped 2 seasons in the step before. Through these features of Detangler, participants were able piece together and triangulate information from each of these views: *"you can check multiple things to make sure you haven't totally broken your code"* (P15).

D. Exploring and understanding data wrangling code

Detangler was able to achieve D1 by providing several ways to explore and understand unfamiliar code. We observe that data scientists frequently toggle transformations (remove and re-add transformation steps) and re-order transformations to form an understanding of the functions applied to the data. Data scientists also expressed the need to refer to function documentation to understand how the data is being modified, which we further discuss in Section VII.

Using Detangler helps validate assumptions about a function's behaviour: We observe that validating one's understanding of code behavior sometimes requires more than reading the documentation for functions, or accessing error logs from its execution. Since participants had varying levels of experience with R, understanding code frequently required interactions with Detangler. While function documentation was useful in situations where participants were confused about unfamiliar functions, it did not help beginners such as P9 or P11 understand how data wrangling operations compose together in relation to the task of creating a visualization. P9 and P11 would often invoke Detangler by clicking on particular lines, and examine the intermediate dataframes at those steps using the code overlay to validate their assumptions about *how the function behaves*. They were able to make guesses based on how the output changed after making a change like reordering lines (P9). In other instances, both P1 and P2 who were quite inexperienced with R were able to guess how `filter` works based on the name and validating their assumption by flipping

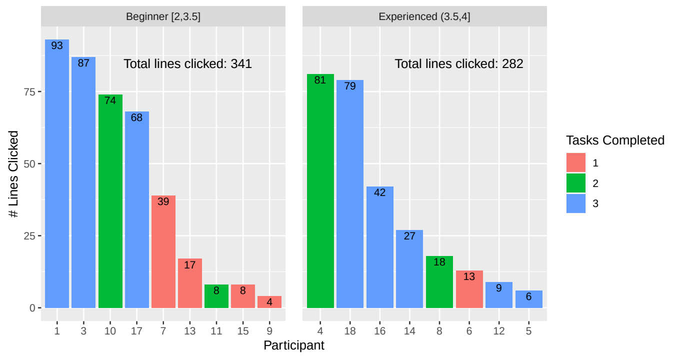


Fig. 7: Number of times participants clicked on transformation steps from the code overlay to view intermediate results.

through the rows of the dataframe in the Table view. We discuss key limitations in supporting novices in Section VII.

E. Detangler aids beginners and experts

Participants generally made heavy use of clicking on lines to invoke Detangler, and focus on particular transformation steps to validate their understanding of the code. While all participants made use of this core invocation and exploration feature of Detangler, we observe that beginners clicked on lines more than experts. This finding was corroborated by the logs as well. Figure 7 visualizes the total lines clicked, and shows that beginners clicked 341 times, whereas experienced participants clicked 282 times. One trend that aligned with this result is that participants who clicked through the intermediate lines of data wrangling code generally completed more tasks besides two exceptions: P11 who was able to complete 2 tasks despite the low number of clicks, and P6 who was having difficulty learning `tidyverse` R style for the first time as a heavy base R user. These log results demonstrate that Detangler successfully supports design goals D1 and D3.

VII. Discussion and Future Work

Detangler's design can be adapted to any fluent code domain. Fluent expressions are used by other popular programming languages, such as LINQ for C# and the pandas and PySpark libraries for Python. We discuss the implications of our findings, future work on improving the interactive exploration of data wrangling code, and identify how Detangler could be adapted to various programming languages and contexts.

A. Better Support for Novice Data Scientists

Although Detangler helped in some ways, it did not provide adequate support participants to complete all tasks. The function documentation was a key requirement in helping several participants understand a function's purpose. Clicking on lines allowed participants to be able to focus on the particular line's code and output. However, as one participant put it, *"[Detangler] was helpful in that I did not have to run every line to view the intermediate data, but I still have difficulty in how to write data wrangling operations and visualizing them."* (P9) While the scaffolded code in our tasks did help novices learn data wrangling in R by examining the example code through Detangler, there was a lack of support in helping them *write* data wrangling code, and be able to understand *why* certain operations are needed for the goal of producing a visualization.

We also observe that beginners refer to function documentation more frequently as compared to experienced data scientists. Through insights from think-aloud sessions, we gather that some functions, like `c`, `drop_na`, `na_if` and `rename` are intuitive. However, the functions `fct_lump` and `fct_reorder` are confusing as *"their function names don't clearly convey its purpose."* These functions, provided by the `forcats` package, operate on categorical variables (called factors) in R. Similarly, `filter` was sometimes confused by participants (P9, P7, P11, P10) to mean *filtering out* rows that meet a criteria, whereas it meant *selecting rows*. Similarly, when clicking on the line using the `pivot_longer` on Task C, P2 was happy to have referenced the documentation: *"Oh values_to! It will be very helpful to have a link to the documentation here."*

B. Live Programming for Data Wrangling

As discussed in Section VI, we found that generally Detangler was able to fit into the workflow of data scientists in RStudio with some pain points. Participants in our study found it useful to be able to highlight data wrangling code and *detangle* it through the Addins and explore within the IDE. However, all participants desired more liveliness once they explored existing code and wanted to make modifications. For example, P10 said *"if I could somehow edit code [in Detangler] that would be cool. Then you can see it updating in real time and you don't have to go back."* P2 expressed how *"having to highlight everything is a little bit weird. It would be really nice if you could do it in the same vein as execute line on editor, it would be a lot more intuitive."* These comments make sense given the current limitations of Detangler. We can take inspiration from tools like Glinda [29] which implement live programming using a domain-specific language for data science programming, or DITL [22], which supports saving snapshots of successful data wrangling code.

C. Exploring and Highlighting Data Quality Issues

Participants leveraged the Data Details view to identify data quality issues, but found a few limitations in the process.

1) Nudging users to validate data issues: During the study, we noted that there was no mechanism to draw attention to check insights available in the Data Details view. For example, P1 who was a beginner said that *"if there were any problems I would like there to be some kind of highlight here like a nudge to click on Data Details."* Experienced participants like P4 kept forgetting to check Data Details and similarly, P10 said *"I think getting used to the fact that each operation has its own data details. Not sure why I kept forgetting, but that's very useful actually."* In other words, while Data Details was useful, it lacked a mechanism to draw attention.

2) Providing flexible tools to audit problems: Our visualizations lacked the ability to provide details-on-demand, meaning participants could identify problems but could not easily locate them. P5, *"One thing that would be super neat to include is what the points in the distribution correspond to"* which is a limitation of the in-lined histograms for each column that others point out as well (P1, P3). For the in-line distribution, P5 and P10 wanted to customize the type of plot to display for the in-line distribution such as a density plot. They also mentioned that showing duplicates of the data would be beneficial but warned that *"it's not easy to handle because it could be duplicates of various columns not just whole row."*, which can be addressed in our future work.

3) Streamlining data quality checks: Many participants (P1, P2, P3, P4, P5, P8, P18) spent time exploring columns that are irrelevant to the task, and yet attracted their attention due to the missing values bar, or potential problems numbers. The always-on visualization and statistics for the columns of a dataframe could become noisy. To mitigate this, P1 desired *"being able to dismiss warnings like dismissing warnings in the IDE."* However, assigning severity to issues and fixing them might always remain a human-in-the-loop effort. The authors of `dataMaid` expressed that automation of cleaning means "all power is given to the the computer with no human supervision, and investigators are less likely to make an active, case-specific choice regarding the handling of the potential errors" [30].

VIII. Conclusion

We built Detangler, a tool that enables structured exploration of data wrangling code and data quality checks using Data Details and Table views. Data scientists must continuously assess and improve data quality by applying and validating chains of transformations—making data wrangling tedious and error-prone. Through formative interviews, we identified challenges with uncovering data quality issues and the manual effort in validating changes, required throughout the data wrangling workflow. In a user study with 18 data scientists, we found that Detangler provides quick and reliable validation of data characteristics using always-on visualizations. Detangler's features help data scientists in (i) identifying potential data smells, (ii) triangulating insights and validating assumptions about data wrangling code, and (iii) effectively debugging wrangling mistakes. Through this work, we aim to be one step closer towards helping data scientists *detangle* their code.

References

[1] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson, "How data science workers work with data: Discovery, capture, curation, design, creation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–15. [Online]. Available: https://doi.org/10.1145/3290605.3300356

[2] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards effective foraging by data scientists to find past analysis choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[3] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–12. [Online]. Available: https://doi.org/10.1145/3313831.3376729

[4] T. Dasu and T. Johnson, *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, 2003, vol. 479.

[5] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson, "How data science workers work with data: Discovery, capture, curation, design, creation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–15.

[6] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[7] M. Fowler and E. Evans, "Fluent interface," *martinfowler.com*, 2005.

[8] M. Fowler, *Domain-specific Languages*. Pearson Education, 2010.

[9] M. B. Kery and B. A. Myers, "Exploring exploratory programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 25–29.

[10] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.

[11] N. Weinman, S. M. Drucker, T. Barik, and R. DeLine, "Fork it: Supporting stateful alternatives in computational notebooks," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–12.

[12] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani, "Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[13] M. B. Kery, D. Ren, F. Hohman, D. Moritz, K. Wongsuphasawat, and K. Patel, "mage: Fluid moves between code and graphical work in computational notebooks," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, 2020, pp. 140–151.

[14] N. Shrestha, T. Barik, and C. Parnin, "Unravel: A fluent code explorer for data wrangling," in *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021, pp. 198–207.

[15] S. Lau, S. S. Srinivasa Ragavan, K. Milne, T. Barik, and A. Sarkar, "Tweakit: Supporting end-user programmers who transmogrify code," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–12.

[16] X. Pu, S. Kross, J. M. Hofman, and D. G. Goldstein, "Datamations: Animated explanations of data analysis pipelines," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.

[17] T. Lieber, J. R. Brandt, and R. C. Miller, "Addressing misconceptions about code with always-on programming visualizations," ser. CHI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 2481–2490. [Online]. Available: https://doi.org/10.1145/2556288.2557409

[18] H. Kang and P. J. Guo, "Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 737–745. [Online]. Available: https://doi.org/10.1145/3126594.3126632

[19] S. Lau, S. Kross, E. Wu, and P. J. Guo, "Teaching data science by visualizing data table transformations: Pandas tutor for python, tidy data tutor for r, and sql tutor," in *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, ser. DataEd '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 50–55. [Online]. Available: https://doi.org/10.1145/3596673.3596972

[20] X. Zhang and P. J. Guo, "Ds.js: Turn any webpage into an example-centric live programming environment for learning data science," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 691–702. [Online]. Available: https://doi.org/10.1145/3126594.3126663

[21] A. Fabri, "pipediff: Show diffs between piped steps," 2022, r package version 0.0.0.9000. [Online]. Available: https://github.com/moodymudskipper/pipediff

[22] A. Y. Wang, W. Epperson, R. A. DeLine, and S. M. Drucker, "Diff in the loop: Supporting data comparison in exploratory data analysis," in *CHI Conference on Human*

Factors in Computing Systems, 2022, pp. 1–10.

[23] I. Rill Data, "Data wrangler," 2022. [Online]. Available: https://github.com/rilldata/rill-developer

[24] B. Greve, *A Beginner's Guide to Clean Data: Practical advice to spot and avoid data quality problems*, 2019.

[25] E. Waring, M. Quinn, A. McNamara, E. Arino de la Rubia, H. Zhu, and S. Ellis, *skimr: Compact and Flexible Summaries of Data*, 2022, r package version 2.1.4. [Online]. Available: https://CRAN.R-project.org/package=skimr

[26] T. Mock, "Tidy tuesday: A weekly data project aimed at the r ecosystem," 2022. [Online]. Available: https://github.com/rfordatascience/tidytuesday

[27] A. Shome, L. Cruz, and A. van Deursen, "Data smells in public datasets," *arXiv preprint arXiv:2203.08007*, 2022.

[28] J. Carlson, "Avoiding traps in member checking," *Qualitative Report*, vol. 15, pp. 1102–1113, 09 2010.

[29] R. A. DeLine, "Glinda: Supporting data science with live programming, guis and a domain-specific language," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–11.

[30] A. H. Petersen and C. T. Ekstrøm, "datamaid: Your assistant for documenting supervised data quality screening in r," *Journal of Statistical Software*, vol. 90, 2019. [Online]. Available: http://statistik-jstat.uibk.ac.at/index.php/jss/article/view/v090i06