Revealing the Semantics of Data Wrangling Scripts With COMANTICS

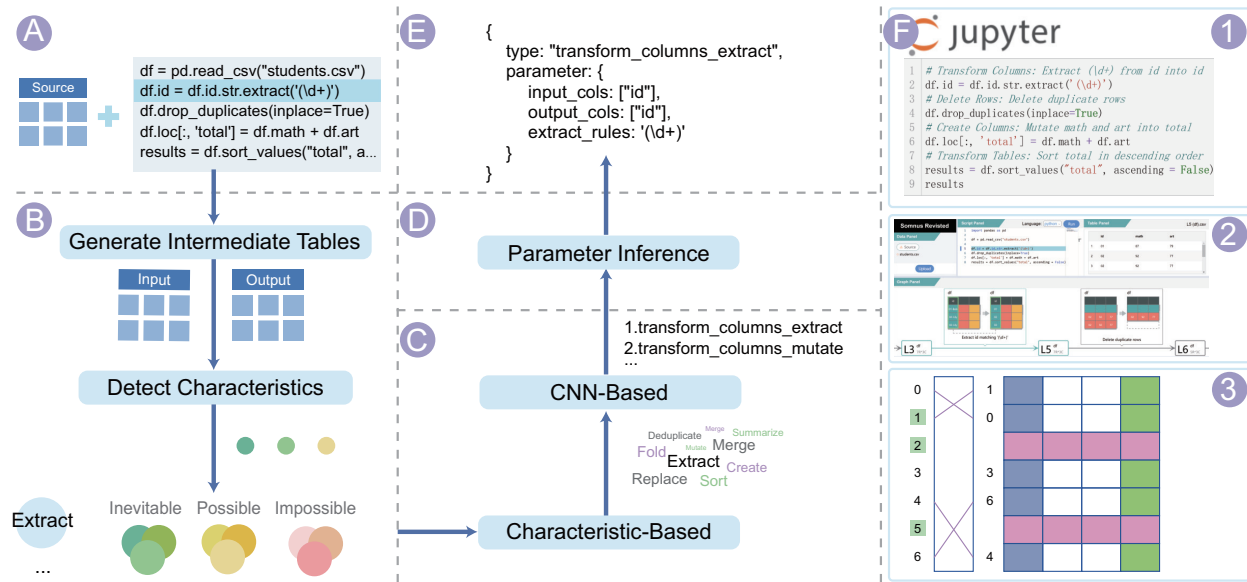Kai Xiong, Zhongsu Luo, Siwei Fu, Yongheng Wang, Mingliang Xu, Yingcai Wu

Fig. 1. COMANTICS is a pipeline that reveals the semantics of wrangling code. (a) shows the input of the pipeline, i.e., a data table and a piece of wrangling script. (b) is the first step of COMANTICS, aiming to generate intermediate tables for each code and detect changes between tables. (c) depicts the second step that identifies transformation type with characteristic-based and CNN-based components. (d) is the third step that infers parameters for the transformation. (e) shows the output of COMANTICS. (f) shows that our pipeline is applied to three applications in different domains.

Abstract— Data workers usually seek to understand the semantics of data wrangling scripts in various scenarios, such as code debugging, reusing, and maintaining. However, the understanding is challenging for novice data workers due to the variety of programming languages, functions, and parameters. Based on the observation that differences between input and output tables highly relate to the type of data transformation, we outline a design space including 103 characteristics to describe table differences. Then, we develop COMANTICS, a three-step pipeline that automatically detects the semantics of data transformation scripts. The first step focuses on the detection of table differences for each line of wrangling code. Second, we incorporate a characteristic-based component and a Siamese convolutional neural network-based component for the detection of transformation types. Third, we derive the parameters of each data transformation by employing a "slot filling" strategy. We design experiments to evaluate the performance of COMANTICS. Further, we assess its flexibility using three example applications in different domains.

Index Terms—Data Transformation, Semantic Inference, Program Understanding, Table Comparison

---

1 INTRODUCTION

- *K. Xiong and Y. Wu are with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China, and with the Zhejiang Lab, Hangzhou, China. E-mail: {kaixiong, ycwu}@zju.edu.cn.*
- *Z. Luo is with Zhejiang University of Technology, Hangzhou, China, and with the Zhejiang Lab, Hangzhou, China. E-mail: rickyluozs@gmail.com.*
- *S. Fu and Y. Wang are with the Zhejiang Lab, Hangzhou, China. E-mail: fusiwei339@gmail.com, wangyh@zhejianglab.com.*
- *M. Xu is with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. E-mail: iexumingliang@zzu.edu.cn.*
- *Yongheng Wang and Siwei Fu are the co-corresponding authors.*

Data wrangling is an arduous and time-consuming routine [13, 17] for data workers who carry out data analysis tasks with different levels of expertise [9, 40]. A commonplace method of wrangling data is writing custom cleaning scripts in various programming languages [29, 34], such as R and Python. In this case, data workers usually seek to understand the semantics of wrangling scripts in various scenarios [70], including debugging for potential code issues, reusing scripts for their data, and maintaining code that is not well-documented. Here, the *semantics of wrangling scripts* means the **type of data transformation** and its **parameters**. For example, given a line of code written in R, `tb2 = arrange(tb1, -num)`, the semantics of the code refers to **sorting** the column **num** of the input table **tb1** in **descending** order, and returning the output table **tb2**.

However, understanding the semantics of intricate scripts requires advanced programming skills and is error-prone [36, 51, 60]. Novice data workers may find it challenging for three reasons. First, data workers need to work across programming languages in some scenarios, such as learning a new programming language and reusing code

written in another language. However, different scripting languages have diverse regulations in implementing data transformations. For example, the *sort* transformation can be implemented by the function `arrange` in dplyr [68] (R), and the descending order is represented by a minus sign or the `desc` function. While in Pandas [53] (Python), the same transformation is implemented by the function `sort_values`, and the descending order is noted as `ascending=False`. Second, in one language, there are different functions to implement the same data transformation. For instance, both `gather` and `pivot_longer` in tidyr [66] (R) can perform the *fold* transformation, and there is a subtle difference in their usage. Third, some functions may omit important parameters for simplicity. These functions either have default parameters (e.g., the `sort_values` function in Pandas performs sorting in ascending order by default), or can extract parameters from the input data tables (e.g., the `left_join` function in dplyr has built-in rules to infer join keys from the two input tables).

A body of prior work has been proposed to understand wrangling scripts. Some tools such as Unravel [57] and WrangleDoc [70] are used for exploring and debugging wrangling scripts. These tools are useful for revealing changes in data tables after performing a data transformation. However, they cannot directly inform data workers of the type of transformation. Others, such as SOMNUS [69], Datamations [50], and Data Tweening [31], illustrate the semantics of data transformations using well-designed visualization. However, they rely on hand-crafted rules to parse scripts, resulting in poor generalizability and scalability.

In this paper, we propose a pipeline, called COMANTICS meaning **co**de se**mantics**, that identifies the semantics of wrangling scripts. We observe that, given a line of wrangling code, changes between input and output tables highly relate to the type of transformation. To have an overview of the changes, we summarize the characteristics of table changes by analyzing a corpus containing 921 lines of real-world wrangling code and present a design space comprising two primary dimensions, i.e., the types of data objects and the properties of data changes. Then, we construct the pipeline consisting of three key steps, i.e., Data Preprocessing, Type Inference, and Parameter Inference. For Data Preprocessing, COMANTICS executes the script and obtains intermediate data tables for each line of wrangling code. Then, the characteristics of table changes are identified automatically. The Type Inference step consists of two components. The first infers transformation type based on the characteristics and obtains a list of candidate transformation types. To identify the transformation with maximum possibility, we employ a Siamese convolutional neural network [8] to rank transformation candidates. Based on the inferred transformation, Parameter Inference aims to identify parameters for transformations. We borrow the "slot filling" strategy and detect parameters based on the wrangling script, table contents, and characteristics. Finally, COMANTICS outputs a data transformation with maximum likelihood and its parameters, which can be applied to a variety of downstream tasks.

We design experiments to evaluate the performance of COMANTICS. We first annotate the real-world corpus with data transformations. Then we assess the generalizability across different programming languages and the contribution of different components of COMANTICS based on five experiment settings. To evaluate the flexibility of our pipeline, we apply it to three applications in different domains, i.e., improving Jupyter Notebook with automatic annotation, augmenting SOMNUS [69] with scalable backend, and enhancing TACO [47] with additional information.

In summary, the contributions of this paper are four-fold: First, we summarize a design space including 103 characteristics of table changes. Second, we construct a novel pipeline, COMANTICS, for inferring semantics, i.e., the types of transformations and their parameters, of wrangling scripts. Third, we evaluate the performance of our pipeline using quantitative experiments. A sub-contribution is that we have built a real-world dataset including 921 lines of wrangling code where each is annotated with a data transformation. Fourth, to assess the flexibility of COMANTICS, we apply our pipeline to three example applications in different domains.

2 RELATED WORK

In this section, we discuss techniques from three aspects, i.e., data wrangling, program understanding in data wrangling, and table comparison.

2.1 Data Wrangling

Data wrangling is a nontrivial activity of cleaning, transforming, and enriching data into the desired form palatable for downstream tasks [17, 29]. Data wrangling often involves a significant number of data transformations [52]. Kasica et al. [30] developed a concise and actionable framework describing multi-table transformation operations.

There are two kinds of approaches for facilitating the process of data wrangling. On the one hand, a plethora of interactive applications has been proposed. Tools for wrangling tabular data [12, 38] like Microsoft Excel, Tableau Prep Builder [58], OpenRefine [23], and Trifacta [64] provide an interactive menu for selecting the desired transformation. Further, Trifacta and its predecessor Data Wrangler [17, 29] integrate an inference engine that generates a ranked list of suggested transformations. Others aim to address wrangling graph [10, 20, 22, 42, 44, 62] and website [7, 24, 39, 41, 46] data. Some of the above tools [17, 23, 29, 58, 64] support recording the process of data transformations through a domain-specific language, and provide textual descriptions of transformations. On the other hand, numerous scripting languages (e.g., Python and R) with data manipulation libraries (e.g., Pandas [53], dplyr [68], tidyr [66], and tidyverse [67]) have been commonly used to wrangle data for their flexibility. However, programming is a complicated and burdensome activity, requiring developers to master specialized skills [36, 48]. To reduce this barrier, increasing research work [15, 16, 19, 26] is motivated by *programming by example* that synthesizes target code given examples provided by data workers. Compared to the aforementioned approaches, our pipeline has different goals, which focus on helping data workers understand wrangling scripts.

2.2 Program Understanding in Data Wrangling

Program understanding in data wrangling is a hot research topic that aims to help data workers comprehend code. Prior work roughly falls into two categories. Firstly, some tools are designed for debugging. For example, WrangleDoc [70] is a JupyterLab extension that leverages program synthesis techniques to automatically generate summaries of code fragments to find subtle bugs. Unravel [57] provides summaries for data transformation functions and an always-on visualization for fluent code to explore and debug the chain of expressions in R. Besides, TweakIt [35] is a system implemented as an extension to Microsoft Excel that applies live interactions to help data workers explore and understand the effects of unfamiliar code. COMANTICS shares the same design goal of facilitating the comprehension of wrangling scripts. However, we focus on the semantics of data wrangling, which consists of the types of data transformations and their parameters.

Secondly, some tools are proposed to visualize the semantics of data wrangling. For instance, Datamations [50] creates animated transitions for data transformations to explain the entire data analysis pipeline. SOMNUS [69] supports transformation inference based on function information and generates a glyph-based provenance graph to visualize the evolution of tables. These visualization tools are useful and effective in depicting the semantics of data processing workflow. However, they rely on a rule-based engine to parse the semantics of the wrangling process, which can hardly scale to different programming languages and a variety of functions. In addition, crafting rules is a laborious process and is limited in real-world applications. COMANTICS, on the other hand, infers the semantics of wrangling scripts from input/output tables and individual lines of code. It is independent of programming languages, libraries, and functions, resulting in better scalability and generalizability. We have successfully applied COMANTICS to three applications (see Sec. 6) to assess its flexibility.

2.3 Table Comparison

Data comparison techniques are widely used to investigate differences between multiple versions of data. Although plenty of work aims to compare time series data, image data, graphs, etc., there are few techniques designed to compare tabular data [47]. Available table

comparison tools, such as ExcelCompare [4], AQT [1], DiffKit [3], Daff [2], Ridom SeqSphere+ [5], and Compare [25], identify and compute differences between two input tables according to metrics including table size differences, cell content differences, the number of unique records and fields, etc., and output statistical results, textual summaries, or visualizations. Sutton et al. [61] presented a tool to detect and explain the differences between two tables and formulate executable patches that can transform one table to be compatible with the other. In addition, Niederer et al. [47] developed TACO, a visual comparison tool for tabular data. It encodes table differences through interactive visualizations to illustrate different types of changes. The aforementioned approaches are successful in comparing differences between tables. COMANTICS goes one step forward in which table differences are identified to infer the semantics of wrangling scripts.

3 A DESIGN SPACE FOR TABLE CHANGES

The semantics of wrangling scripts consists of transformation types and their parameters. According to Xiong et al. [69], identifying the transformation type is the key challenge. We observe that differences between input and output tables of a line of wrangling code can reveal the transformation type to some extent. For example, if some rows are deleted while the others remain unchanged, the transformation seems to belong to *delete rows*. Further, if all deleted rows contain missing values while the remaining do not, the transformation type is likely to be *delete rows with missing values*. Motivated by this observation, we explore the characteristics of table changes and present a design space to understand how a table is changed based on data transformations.

3.1 Methodology

To understand the characteristics of table changes, we collect real-world corpus and analyze how wrangling scripts transform input data tables into outputs. The corpus is collected from GitHub[1] and Kaggle[2]. We use keywords such as data wrangling and data cleaning to find target scripts. Furthermore, we collect corpus from prior work [16, 30]. The collection procedure is based on two inclusion criteria. First, as our approach relies on table changes, we focus on scripts with data tables. Second, due to our expertise in programming, we keep scripts written in Python and R, and balance the number of scripts in both languages. After collecting the corpus, we manually "clean" these scripts to meet the requirement of COMANTICS. Besides data wrangling, the scripts usually include code for other purposes, such as visualization and machine learning. We carefully remove these code and ensure that the remaining can run and process the input data correctly. Then, we deal with function chaining, where multiple functions are executed consecutively on the same variable. An example is shown in Fig. 2(a), four functions (i.e., `filter`, `mutate`, `select`, and `arrange`) are called in sequence on the dataset `mtcars`. This language pattern is efficient for developers. However, it is not supported in COMANTICS because we can hardly obtain intermediate tables. Hence, we rewrite chained functions into individual lines of transformations (Fig. 2(b)). Finally, our corpus contains 74 curated scripts, where 42 of them are written in Python and 32 are in R, adding up to 921 lines of wrangling code.

```
1  auto_cars <- filter(mtcars, am = 0) %>%
2      mutate(sec = qsec * 4) %>%
3      select(drat, sec) %>%
4      arrange(-sec)                        Ⓐ
```

```
1  auto_cars <- filter(mtcars, am = 0)
2  auto_cars <- mutate(auto_cars, sec = qsec * 4)
3  auto_cars <- select(auto_cars, drat, sec)
4  auto_cars <- arrange(auto_cars, -sec)    Ⓑ
```

Fig. 2. An example of unraveling (a) a chained syntax into (b) a sequence of statements in the R programming language.

To code the characteristics of table changes, we first execute these scripts to obtain the intermediate tables for each line of wrangling code. Then we record the differences between tables in detail from various aspects as much as possible. Next, we apply a qualitative method, open coding and axial coding rooted in grounded theory, to these table differences. Specifically, we label each difference with a descriptive code and draw connections between these codes. Based on our observations and wrangling experiences, we adopt the codes which are conducive to distinguishing between different transformations. After frequent discussions with two data scientists, we group and condense related codes into broader categories. Finally, we derive five properties of data changes and four data objects from these categories.

3.2 Dimensions of the Design Space

By analyzing the characteristics of table changes, we construct the design space consisting of two primary dimensions, i.e., data objects and the property of data changes. Data objects include four different types, i.e., Tables, Columns, Rows, and Cells. The first three data objects are recognized in various prior work in data wrangling [30, 69] and table comparison [47]. We introduce Cells additionally because it relates to a wide range of transformations including editing the text of a cell and filling empty cells with adjacent cells. The second dimension is the property of data change. We identify five high-level properties, i.e., Number, Order, Relation, Value, and Type. Intersections between data objects and properties mean a set of characteristics, which are described by inequalities. For example, the intersection between Tables and Number includes five characteristics, i.e., the number of input and output tables goes from "0 → 1", "1 → 0", "1 → 1", "1 → many", and "many → 1", as illustrated in Fig. 3. We notice that the mapping between data objects and properties is not complete, meaning that some properties are only applicable to specific data objects. In the following paragraphs, we describe the five properties in detail.

3.2.1 Number

Numerous transformations can result in a change in the number of data objects including *create tables*, *delete columns*, etc. Number is applicable to tables, columns, and rows, as shown in Fig. 3, For tables, the comparison between the number of inputs and outputs is critical. For example, the *create tables* transformation may increase the number of tables from 0 to 1. Meanwhile, the *separate tables* may generate more tables based on one input table. Therefore, we include five characteristics in this category, i.e., the number of input and output tables goes from "0 → 1", "1 → 0", "1 → 1", "1 → many", and "many → 1", which corresponds to characteristics 1—5 in Fig. 3.

For columns and rows, we summarize characteristics based on single- or multiple-table operations. In terms of single-table transformations, we focus on whether the number of columns/rows in the output table is greater than (i.e., >, the 8th characteristic in Fig. 3), equal to (=), or smaller than (<) the number in the input table. In terms of multiple-table transformations, changes in the number of columns/rows may vary. For example, considering the *inner join* transformation, the number of rows in the output table is smaller than or equal to the minimum number of rows in the input tables. On the contrary, the *split table by groups* transformation may split a table row-wise. And the number of rows in the input table equals the sum of rows in the output tables. For detailed categorization, we further divide multiple-table transformations into two cases, i.e., 1 → many (one input table and many output tables) and many → 1. For the 1 → many case, we compute the minimum, maximum, and sum of the number of columns/rows in all output tables, and compare them to the number in the input table (e.g., characteristics 9—15). For the many → 1 case, the same computation is performed for input tables, which is compared to the number of outputs (e.g., characteristics 16—22). Each case includes seven characteristics, which are described in Fig. 3.

3.2.2 Order

The order property describes the position change of columns and rows, which can be performed manually or be sorted according to criteria including disorder, ascending, and descending [30]. When performed

Outputs : ■   Inputs : ■

	Table	Column	Row	Cell
Number	1. ■ > 0 & ■ = 0 2. ■ = 0 & ■ > 0 3. ■ > 0 & ■ = ■ 4. ■ > 0 & ■ > ■ 5. ■ > 0 & ■ < ■	6. ■ = ■ 7. ■ < ■ 8. ■ > ■ 9. min (■) > ■ 10. min (■) = ■ 11. min (■) < ■ & max (■) > ■ 12. max (■) = ■ 13. max (■) < ■ & sum (■) > ■ 14. sum (■) = ■ 15. sum (■) < ■ 16. ■ < min (■) 17. ■ = min (■) 18. ■ > min (■) & ■ < max (■) 19. ■ = max (■) 20. ■ > max (■) & ■ < sum (■) 21. ■ = sum (■) 22. ■ > sum (■)	23. ■ = ■ 24. ■ < ■ 25. ■ > ■ 26. min (■) > ■ 27. min (■) = ■ 28. min (■) < ■ & max (■) > ■ 29. max (■) = ■ 30. max (■) < ■ & sum (■) > ■ 31. sum (■) = ■ 32. sum (■) < ■ 33. ■ < min (■) 34. ■ = min (■) 35. ■ > min (■) & ■ < max (■) 36. ■ = max (■) 37. ■ > max (■) & ■ < sum (■) 38. ■ = sum (■) 39. ■ > sum (■)	⊘
Order	⊘	40. index(■) = index(■) 41. index(■) ≠ index(■) 42. order(■) = disorder 43. order(■) = ascend 44. order(■) = descend	45. index(■) = index(■) 46. index(■) ≠ index(■) 47. order(■) = disorder 48. order(■) = ascend 49. order(■) = descend	⊘
Relation	⊘	50. ■ ⊂ ■ 51. ■ ⊃ ■ 52. ■ = ■ 53. ■ ≠ ■ 54. ¬∃ duplicate(■) & ∃ duplicate(■) 55. ∃ duplicate(■) & ¬∃ duplicate(■) 56. ¬∃ duplicate(■) & ¬∃ duplicate(■) 57. ∃ duplicate(■) & ∃ duplicate(■)	58. ■ ⊂ ■ 59. ■ ⊃ ■ 60. ■ = ■ 61. ■ ≠ ■ 62. ¬∃ duplicate(■) & ∃ duplicate(■) 63. ∃ duplicate(■) & ¬∃ duplicate(■) 64. ¬∃ duplicate(■) & ¬∃ duplicate(■) 65. ∃ duplicate(■) & ∃ duplicate(■)	66. ■ ⊂ ■ 67. ■ ⊃ ■ 68. ■ = ■ 69. ■ ≠ ■ *same colum* 70. ¬∃ duplicate(■) & ∃ duplicate(■) 71. ∃ duplicate(■) & ¬∃ duplicate(■) 72. ¬∃ duplicate(■) & ¬∃ duplicate(■) 73. ∃ duplicate(■) & ∃ duplicate(■) *same row* 74. ¬∃ duplicate(■) & ∃ duplicate(■) 75. ∃ duplicate(■) & ¬∃ duplicate(■) 76. ¬∃ duplicate(■) & ¬∃ duplicate(■) 77. ∃ duplicate(■) & ∃ duplicate(■)
Value	78. ¬∃ missing (■) & ∃ missing (■) 79. ∃ missing (■) & ¬∃ missing (■) 80. ¬∃ missing (■) & ¬∃ missing (■) 81. ∃ missing (■) & ∃ missing (■) 82. ■ ⊇ specific val 83. ■ ⊉ specific val	84. ¬∃ missing (■) & ∃ missing (■) 85. ∃ missing (■) & ¬∃ missing (■) 86. ¬∃ missing (■) & ¬∃ missing (■) 87. ∃ missing (■) & ∃ missing (■) 88. ■ ⊇ specific val 89. ■ ⊉ specific val	90. ¬∃ missing (■) & ∃ missing (■) 91. ∃ missing (■) & ¬∃ missing (■) 92. ¬∃ missing (■) & ¬∃ missing (■) 93. ∃ missing (■) & ∃ missing (■) 94. ■ ⊇ specific val 95. ■ ⊉ specific val	96. ! missing (■) & missing (■) 97. missing (■) & ! missing (■) 98. ! missing (■) & ! missing (■) 99. missing (■) & missing (■) 100. ■ = val1 & ■ = val2 101. ■ ≠ val1 & ■ = val2
Type	⊘	102. type (■) = type (■) 103. type (■) ≠ type (■)	⊘	⊘

Fig. 3. The design space describes characteristics of table changes that consist of two dimensions, i.e., data objects and properties of data changes. 103 characteristics are included in the space. The blue and orange square indicates data objects in the output and input table, respectively. A detailed description of each characteristic is provided in the supplemental material.

manually, the index of columns/rows is changed (e.g., characteristics 40 and 41). When sorted based on criteria, the content of the output column/row may be ordered (e.g., characteristics 42—44).

3.2.3 Relation

Some data transformations involve relations between data objects (e.g., columns, rows, and cells). Taking the `unite` function in tidyr as an example, `df2 = unite(df1,'Z', X, Y)` forms the column `Z` in `df2` by concatenating strings of columns `X` and `Y` in `df1`. Relations can be triggered by mathematical formulas and functions. However, the relation space between data objects is too large to be exhausted. By analyzing real-world wrangling scripts, we focus on four commonly-used set relations, i.e., subset ($\subset$), superset ($\supset$), equal ($=$), and others (i.e., apart from these three relations, $\neq$).

For relations between columns (or rows), we summarize within- and between-table characteristics, respectively. In terms of between-table characteristics, we focus on whether the content of columns in the input table is a subset ($\subset$), superset ($\supset$), identical ($=$), or others ($\neq$) to that of the output table (e.g., characteristics 50—53). In terms of within-table characteristics, we mainly focus on identical ($=$) relations. That is, we first identify the existence of identical columns within input and output tables, and compare them using boolean operators (e.g., characteristics 54—57). For relations between cells, the characteristics are described by whether the values are equal ($=$), substring (e.g., $\subset$ and $\supset$), or others ($\neq$) to each other (e.g., characteristics 66—69). Besides, we check whether cells in the same column or row have identical values in the input and output tables (e.g., characteristics 70—77).

3.2.4 Value

Some transformations relate to special values, e.g., missing values and user-defined values. For example, the `dropna` function in Pandas detects and removes rows with missing values. Meanwhile, the `replace` function replaces cells with user-defined values. The value property is applicable to all data objects. For tables, columns, and rows, we establish characteristics by identifying the existence of missing values within input and output data objects, and compare them using boolean operators (e.g., characteristics 78—81). In addition, we detect whether the data objects include ($\supseteq$) a user-defined value or not ($\not\supseteq$). As for cells, the characteristics are described by whether a cell is a missing value, and whether a cell has changed from one value to another.

3.2.5 Type

This property, which describes changes in data type, is applicable to columns. We focus on three types of columns, i.e., nominal, quantitative, and temporal. The characteristics are described as, given a column in the input table, whether its data type is changed in the output table.

3.3 Inferring Transformations With Characteristics

We summarize the characteristic space for the purpose of inferring the type of data transformation. We begin with collecting a set of transformations. Then we discuss the principles of type inference.

Kasica et al. [30] presented a design space encompassing 15 high-level transformations, which provides a good start for our investigation. However, some transformations are too rough to distinguish detailed semantics. For example, constructing columns manually, mutating values from existing columns, and merging multiple columns, are all summarized by *Create Columns* without distinction. On the other hand, Wrangler [17] and Trifacta [64] provide vocabularies describing low-level operations, such as *Extract*, *Merge*, and *Replace*. To support finer-grained semantics, we manually annotate each line of code in our corpus (see Sec. 3.1) by linking one high-level transformation and one low-level operation. Taking the above three transformations as examples, the new vocabularies of them are *create_columns_create*, *create_columns_mutate*, and *create_columns_merge*, respectively. This strategy is effective in generating a large number of low-level transformations. In addition, we observe that some wrangling code does not change data tables, such as `group`, `ungroup`, and `reset_index` functions in Pandas. Though not changing data tables immediately, these functions usually act as prerequisites for the following transformations.

Hence, we include an *identical_operation* transformation to describe these functions.

To infer transformation type, we establish a mapping between transformations and characteristics. For each transformation, we categorize characteristics into three groups, i.e., impossible, possible, and inevitable. Impossible characteristics refer to those that should not appear when performing a transformation. Inevitable characteristics, on the other hand, will definitely exist after a transformation. The above two sets play major roles in characteristic-based inference (see Sec. 4.2.1). Possible characteristics are those that may or may not appear. For example, given a line of code, `df=df[df.num>1]`, which removes rows not meeting the condition `df.num>1`, the number of columns will not change (e.g., the characteristic 6 is inevitable), and the number of rows in the output table can be smaller or equal to that in the input table (e.g., characteristics 23 and 24 are possible). However, changes in the number of columns (e.g., characteristics 7 and 8) are impossible. As a result, the problem of inferring transformation type is turned into detecting a set of impossible, possible, and inevitable characteristics.

4 THE COMANTICS PIPELINE

This section introduces COMANTICS, an automatic pipeline for inferring semantics for data wrangling scripts. Our pipeline takes data tables and a wrangling script as input, and generates a sequence of data transformations with parameters. As illustrated in Fig. 1, COMANTICS consists of three key steps, i.e., Data Preprocessing (Fig. 1(b)), Type Inference (Fig. 1(c)), and Parameter Inference (Fig. 1(d)).

4.1 Data Preprocessing

Since table changes highly relate to the semantics of transformation, the goal of this step is to identify a set of characteristics of table changes. To this end, we first execute the wrangling script on the given source table, and obtain intermediate input and output tables for each line of code, which are saved as CSV files for further analysis. These input tables are critical for type inference as one line of wrangling code could have different semantics given different input tables [69]. Second, we detect table changes based on the design space for each wrangling code and output a set of characteristics. We notice that the detection is time-consuming since some characteristics require content-based comparison between two tables. For example, detecting the relation of two columns involves a pair-wise comparison of the content in each cell. Therefore, we employ two pruning strategies to facilitate the detection.

Characteristics Pruning: Most detection relies on the existence of input and output tables. Hence, a number of characteristics can be pruned if the prerequisite does not stand. For example, if only output tables exist, the transformation is likely to be *Create Tables*, and the detection of other characteristics, such as changes in order and relation, is not applicable.

Data Objects Pruning: We observe that, given new data objects in the output table, characteristics are mainly caused by the new ones. Hence, we prune the remaining objects for content-based comparison. For example, after detecting new columns in the output table, we focus on the detection of characteristics based on the new columns. Other columns in the output tables, however, will not be involved in the content-based comparison.

4.2 Type Inference

To infer the transformation type of a line of wrangling code, the Type Inference step includes two components, i.e., characteristic-based inference and model-based inference. As the space of transformations could be large, the first component filters out impossible transformations and results in a smaller number of candidates. The second component utilizes a Siamese convolutional neural network to assess the possibility of each candidate and returns a sorted list based on possibilities.

4.2.1 Characteristic-Based Inference

Given a collection of characteristics, we filter out impossible transformations based on the mapping described in Sec. 3.3. We establish two main strategies. First, we identify whether characteristics fall into the *impossible* set of each transformation. If true, the transformation is not

Fig. 4. Examples of the two ambiguities in characteristic-based inference. (a) The output table is identical to the input table, which can be caused by *Remove Duplicate Rows*, *Remove Rows with Missing Values*, *Sort Age in Ascending Order*, etc. (b) The second row has been deleted, which can be caused by performing *Filter Rows Where Gender is Male*, *Remove Rows with Missing Values*, etc.

included in the candidate list. Second, we filter out transformations whose *inevitable* set is not a subset of the set of characteristics. In this component, the type of transformation can hardly be uniquely identified because of two kinds of ambiguities. First, the output table may be identical to the input one. This may be because the conditions of transformations do not meet. An example is shown in Fig. 4(a), the input table may be unchanged after *Remove Duplicate Rows* because no duplicates exist in the input. However, the same result may be caused by *Remove Rows with Missing Values* or *Sort Age in Ascending Order*, etc. Second, more than two transformations lead to the same changes between tables. As Fig. 4(b) shows, changes between two tables may be caused by *Filter Rows Where Gender is Male*, *Remove Rows with Missing Values*, etc. Hence, this component results in a set of transformation candidates.

4.2.2 Model-Based Inference

To resolve the ambiguities in characteristic-based inference, the analysis of scripting code seems a feasible solution. However, due to the variety of programming languages, functions, and parameters in data wrangling, crafting rules to parse code is time-consuming and limited in generalizability and scalability. Besides table changes, we observe that the semantics of a transformation is usually implied in the code. For example, some functions, e.g., `drop_duplicates` in Pandas, `distinct` in dplyr, and `unique` in R, perform *Removing Duplicate Rows*, which can be inferred from the function name to some extent. It is also true for some parameters, such as `ascending=False`, `remove=TRUE`, etc. Therefore, we employ a machine learning model to alleviate ambiguities and compute the possibilities for each transformation candidate. One key step in machine learning is feature engineering, which identifies critical information and generates vectors for it. Here we incorporate the characteristics, function names, and parameters as key information for feature engineering.

To reveal the semantics of function names and parameters, we vectorize them using text embedding approaches. The embedding, however, is not straightforward. To be specific, some wrangling code does not include a function name. Fig. 6(c) presents an example where a new column in the output table is created by an addition statement. In this case, we set `None` as its function name. In addition, function names and parameters usually contain out-of-vocabulary (OOV) words. For example, some functions are named `drop_duplicates`, `inner_join`, and `fillna`, and some parameters are called `by_group`, `to_replace`, and `skipna`. To tackle this problem, we adopt FastText [11] for text embedding because it can handle the OOV words by considering subword information. Finally, we obtain two 1×300 vectors capturing the semantics of function names and parameters, respectively. Similarly, we present each characteristic using natural language, and use FastText to vectorize each characteristic. After obtaining all feature vectors of characteristics, we average all vectors to form one with dimension 1×300. Finally, we linearly combine these vectors to obtain a semantic vector with dimension 3×300.

To calculate the likelihood of each transformation, we employ a Siamese Convolutional Neural Network (CNN) [8] which can address two challenges. First, the problem of inferring transformation type is relatively new, and training samples are hard to find in the existing corpus. Second, the data transformation space could be large. However, only a few of them are used commonly in practice (details are described

in Sec. 5.1). That means training samples from the real-world corpus are likely to be imbalanced. CNN is a kind of flexible classification architecture that is suitable for few shot problems and imbalanced data [33]. We acknowledge that COMANTICS does not rely on one model architecture, and the CNN model can be replaced by others. In this paper, we use CNN to verify the idea of COMANTICS.

The Siamese network model comprises two identical subnetworks, and each utilizes five convolutional layers followed by a flatten layer. The input of each subnetwork is a semantic vector encompassing three types of information, i.e., the characteristics of changes, the function name, and the function parameters. In this model, we refer to Hadsell et al. [18] to define and optimize the contrastive loss. After a pair of semantic vectors is fed into the model, the similarity of the two vectors is computed based on Euclidean distance. The output of the model is 1 or 0, which means whether the two inputs belong to the same transformation type. After training the model on the labeled dataset (see Sec. 5.1), we apply it to obtain the likelihood of each transformation in the candidate set.

4.3 Parameter Inference

Parameters define how a transformation performs. Taking the set of transformation candidates with likelihood as input, this step aims to infer parameters for transformations. Borrowing the idea of slot filling in task-based dialog systems [56], we first manually define a set of parameters, i.e., slots, that a transformation should include. Then, we detect these parameters, i.e., filling slots, based on the wrangling code, characteristics of table changes, and the content of input/output tables. For example, given a piece of code, `df2 = unite(df1, 'Z', X, Y)`, COMANTICS infers that the transformation *merge columns* has the maximum likelihood. Then, we detect parameters of *merge columns*, which are pre-defined as one input table, multiple columns in the input, one output table, the merged column in the output, and a separator. The input and output tables can be obtained from the Data Preprocessing step. To fill other slots, we first extract column names from the code, i.e., `Z`, `X`, and `Y`. Then, we obtain relations between the three columns using characteristics of table differences, and fill the two slots, i.e., multiple columns in the input and the merged column in the output. Since the separator is not available in the code, we extract it from the table content. Note that COMANTICS only supports the inference of a limited number of separators, and can be easily extended.

However, the slot filling process may fail. If so, COMANTICS will filter out the transformation and turn to the one with the second maximum likelihood and detect whether parameters meet the transformation, and so forth. COMANTICS stops detection until parameters match the transformation type. Following the above example, if relations between columns cannot be detected, the transformation type, e.g., *merge*, does not stand. Then, COMANTICS will turn to the next transformation and continue the slot filling process. This step can also filter out impossible transformations to some extent, which is beneficial to the overall accuracy of COMANTICS.

5 EXPERIMENTS

We design experiments to evaluate COMANTICS from two aspects. First, since characteristic-based inference is independent of programming languages, we assess whether the pipeline trained in one programming language can be generalized to the other. Second, we evaluate how each part of COMANTICS contributes to the overall performance.

Table 1. Statistical analysis of the dataset on five metrics, including the total lines of wrangling code (#Instances), the average lines of wrangling code in scripts (Avg #Inst), the number of distinct data transformation types (#Trans), the number of distinct functions (#Func), and the average number of implementations for each data transformation (Avg #Imps).

	#Instances	Avg #Inst	#Trans	#Func	Avg #Imps
Python	606	14.43	29	52	3.86
R	315	9.84	28	47	2.14
Sum	921		30	99	

5.1 Dataset

Based on the corpus described in Sec. 3.1, we manually annotate the transformation type for each line of code. To ensure the accuracy, the labeling process is based on the documentation of each function describing its usage. After that, we verify the labeling results using the characteristics between the input and output tables. Finally, we have labeled and verified 74 script files including 921 lines of code. We distinguish scripts written in Python and R, and report the statistical results of our dataset on five metrics (see Table 1). Here we define the number of implementations (#Imps) as how many methods (including different functions or the non-functional equation) can be utilized to perform a data transformation. For example, there are two functions (i.e., `pivot_longer` and `gather`) in tidyr that can be used to fulfil *transform_tables_fold*, so its #Imps is 2.

From Table 1, we have some interesting findings. Though we have tried to balance the number of scripts (42 for Python and 32 for R), the number of instances between the two languages varies (606 versus 315). It is because Python scripts are usually longer than those of R (14.43 Avg #Inst for Python versus 9.84 for R). The number of transformations and functions that appeared in the corpus is similar. COMANTICS can support 99 functions in total, which is significantly more than previous work including Datamations [50] and Sonmus [69]. We argue that our pipeline is scalable in terms of the number of functions. However, we notice that the Avg #Imps in Python (3.86) is greater than that in R (2.14), meaning that Python is more flexible in implementing data transformations. The dataset and its statistical analysis are provided in the supplemental material.

5.2 Metrics

Following the previous work [55], we adopt Top-N accuracy metrics to evaluate the performance of COMANTICS. These metrics are usually used for multi-class classification models. Top-1 accuracy is a conventional classification metric that measures the proportion of samples where the highest probability prediction matches the target label. However, due to a large number of categories (i.e., 30 distinct transformations) and a few training/testing samples (i.e., 921 instances), Top-1 accuracy may limit our understanding of COMANTICS. Hence, we employ Top-3 accuracy, which measures the possibility that the top three predictions include the correct answer. We also report the training times in minutes for each experiment to get a rough idea about how computationally-demanding the training process is.

5.3 Methods and Apparatus

To evaluate how each component contributes to the performance, we prepare two techniques, i.e., the entire pipeline and the Siamese Convolutional Neural Network (CNN). Similar to COMANTICS, the CNN technique includes three steps. The difference between them is that the second step of the CNN technique does not include a characteristic-based inference component.

We split the dataset using four strategies. Firstly, to evaluate how each technique performs across Python and R, we use the scripts in Python as the train set and R as the test set, and vice versa. To ensure that the space of data transformations is consistent between the train and test, we select common transformations (27 types in total) in both sets. Secondly, we split the Python instances by random selection in each transformation type at a ratio of 80 : 20, which are used as the train and test sets, respectively. Based on the Python test set, we form the third strategy where we include R scripts into the train set. Finally, we combine all Python and R instances and randomly split them at a ratio of 80 : 20. The experiments are performed on a Windows 10 desktop with a 3.20GHz Intel Core i7-8700 CPU and 32 GB RAM.

5.4 Results and Analysis

Contribution of each component: The Type Inference step of COMANTICS consists of two components, i.e., characteristic-based inference and CNN-based inference. By incorporating the characteristic-based inference component, the pipeline results in better accuracy compared to the CNN-based model on both Top-1 and Top-3 accuracies. Specifically, based on the Top-1 accuracy, the accuracy increases by 21.32%

Table 2. The training time, Top-1, and Top-3 performances of COMANTICS and its CNN-based component in different experiment settings.

	Setting		CNN		COMANTICS		Time
	Train	Test	Top-1	Top-3	Top-1	Top-3	(minutes)
1st	Python	R	19.1	30.3	53.8	76.1	19.5
2nd	R	Python	23.2	32.3	44.5	82.6	15.1
3rd	Python	Python	62.1	83.3	80.3	94.7	18.3
4th	All	Python	72.0	90.9	90.2	98.5	21.9
5th	All	All	78.0	92.2	92.2	99.0	19.6

on average ($\sigma = 7.06$) for the five data splits. It is also notable that CNN-based inference achieves acceptable performance for the last three data splits, indicating that both components have a great contribution to the pipeline.

Generalizability to different languages: To understand the generalizability, we focus on the first two experiment settings. Generalization across Python and R is a hard task. COMANTICS achieves 53.8% and 44.5% Top-1 accuracy for Python-R and R-Python, respectively, while the accuracy of CNN-based model is lower, merely 19.1% and 23.2%, respectively. We speculate the low accuracy is due to the small number of train samples which are not enough for the model to capture features.

Rationality of semantic vector: By comparing the results of the 3rd and 4th data splits, we infer that the embedding of function names and parameters is significant in type inference. Specifically, by incorporating R scripts into the train set, the semantics in R has joined for inference, which improves the Top-1 accuracy of both CNN-based model and COMANTICS by 9.9%. This finding inspires us for improving the pipeline design in future research, where we plan to include synonyms for each transformation in model training.

Analysis of Confusion Matrix: Referring to the normalized confusion matrix (provided in the supplemental material), we observe that *create_rows_summarize* (i.e., creating rows by summarizing other rows with aggregate operations such as mean, sum, and max) tends to be predicted as *create_rows_create*, which means rows are created manually. Another case is that some *transform_columns_extract* (i.e., transforming columns by extracting values from one column) are classified as *transform_columns_mutate*. We infer two reasons for the two cases. First, characteristics in our design space cannot describe complex relations such as summarization and regular expressions. Second, the instances of these transformations are few, making it difficult for COMANTICS to learn their features.

5.5 Potential Improvements

Although COMANTICS has achieved 92.2% of the Top-1 accuracy when combining R and Python scripts in the train and test sets, the analysis of failure cases informs us of potential improvements in our pipeline.

First, the semantics of parameters should be investigated by advanced approaches. Currently, we use FastText to convert all parameters into one feature vector, which may overlook key semantics. For example, by analyzing a failure case, `df4 = filter(df3,!is.na(ST))`, it should be classified as *delete_rows_dropna*, while COMANTICS incorrectly predicted it as *delete_rows_filter*. This case is "difficult" because the two transformations are semantically similar, i.e., deleting rows with missing value is a kind of deleting rows using filtering conditions. By analyzing the code, we note that the parameter `!is.na(ST)` is a key factor to distinguish it from *delete_rows_filter*. The failure case indicates that the Type Inference step of COMANTICS has not fully revealed the semantics of parameters. We plan to investigate natural language processing techniques such as tokenization [14, 27, 43, 65] to address this issue.

Second, characteristics detection can be further augmented. The characteristic-based inference component is powerful in improving the performance of the pipeline. However, given a pair of input and output tables, characteristics detected in the current prototype are sometimes ambiguous for type inference (as described in Fig. 4). We observe that the detection of characteristics highly relies on the input data tables. Thus, to resolve ambiguity, we plan to employ well-crafted data tables to verify each transformation candidate. For example, if

delete_rows_deduplicate is one of the candidates, a well-designed input table with duplicate rows would be constructed to check whether *delete_rows_deduplicate* stands for the line of code.

6 EXAMPLE APPLICATIONS

In this section, we demonstrate the flexibility of COMANTICS through three applications in different domains. First, we show how COMANTICS supports semantic annotation for scripts in Jupyter Notebooks. Second, we integrate COMANTICS into an existing program visualization system to augment its ability to parse wrangling scripts. Third, we demonstrate an idea of improving an existing visualization system with additional information obtained from COMANTICS.

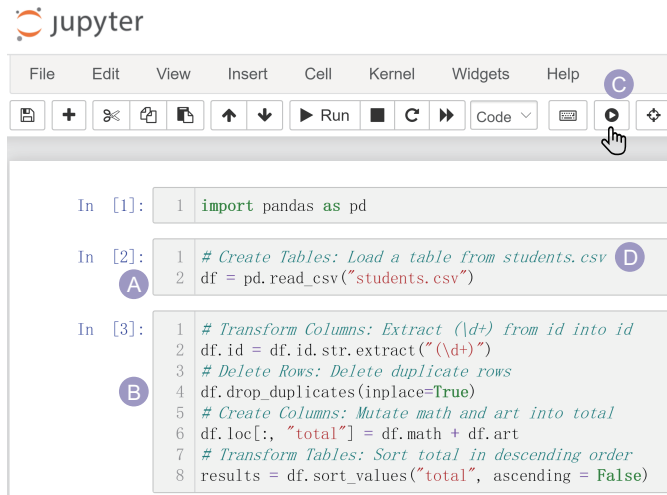### 6.1 Code Annotation in Jupyter Notebook

Fig. 5. An instance of generating annotations for data wrangling code in Jupyter Notebook. After wrangling code has been written in two code cells (a, b), the annotations (d) describing transformation semantics are appended to the original code by clicking the extension button (c).

Computational notebooks leverage an interactive literate programming [32] paradigm that combines code, natural language text, and execution results of code cells in a single document. Nowadays, increasing data workers adopt them for exploratory data analysis and sharing of computational narratives [49]. However, writing comments for each line of code could be burdensome. To assist the processing of documenting for data workers, we implement COMANTICS as a Jupyter Notebook extension to generate semantic annotations for wrangling code snippets. Specifically, to obtain intermediate input and output tables, we utilize the Variable Inspector [6] to collect all variables with their types, sizes, and values. After COMANTICS infers the transformation type and its parameters of a line of wrangling code, the extension employs a template-based approach to generate a natural language sentence describing the semantics. And the sentence will be appended above to the original code as a comment. As presented in Fig. 5(a) and (b), there are five lines of wrangling code in the script, which are distributed in two different code cells. After clicking the extension button (Fig. 5(c)), comments will be automatically inserted above each line of code (Fig. 5(d)). As code annotation is a hot topic in the domain of software engineering [21,37], COMANTICS shows potential for code annotation in data wrangling.

6.2 SOMNUS Re-visited

COMANTICS can also be used to expand the capacity of existing visualization systems that reveal the semantics of transformations. For example, SOMNUS [69] is a program visualization system that generates a provenance graph to visualize a script of data wrangling. However, it relies on hand-crafted rules to parse code and infer transformation types. Hence, the backend of the system is limited in scalability. To augment SOMNUS, COMANTICS replaces its Program Adaptor module that parses code and infers the type of transformation. As a result, SOMNUS supports a larger number of functions, and more types of statements (e.g., non-functional assignments and non-assignment statements). Fig. 6 shows three examples that SOMNUS can support with the help of COMANTICS. In Fig. 6(a), the `extract` function was not supported in SOMNUS, and establishing rules for new functions is time-consuming. Fig. 6(b) shows that the non-assignment statement was not supported because the input and output table is not explicitly presented. Fig. 6(c) depicts a non-functional assignment that can be hardly mapped to manually-crafted rules. Our corpus (see Sec. 3.1) shows that 507 out of 921 (55.05%) instances match the above three scenarios. By incorporating COMANTICS, SOMNUS extends its ability to deal with a wider range of functions and statements.

6.3 TACO Re-visited

TACO [47] is a visual analytics system that facilitates table comparison. It presented four types of changes between two tables including 1) *structural changes* where rows or columns are added or removed, 2) *content changes* where the values of cells are modified, 3) *reorder changes* where rows or columns are shifted to other positions, and 4) *merge/split changes* where multiple rows or columns are combined into a single one or vice versa. Though similar, our design space is the superset. That is, our space includes characteristics that are not supported in TACO, such as inclusion relationships between columns/rows (e.g., characteristics 50—53), ordering states (e.g., characteristics 42—44), and data types (e.g., characteristics 102 and 103). Therefore, COMANTICS can assist TACO to reveal more characteristics of table changes.

We express this idea using a simplified but extended version of TACO. In this version, we use additional visual channels to represent two relationships, i.e., green markers on the left side encode duplicate rows, and brown markers on top of the heatmap to display the superset relations between two columns. Fig. 7 shows an example. Based on the original visual encodings, we observe two red rows, which indicate that they are deleted. However, it is not clear why they are removed. With the help of COMANTICS, we can infer the reason with relation information between columns and rows. For example, we can observe that there are three green markers on the left side, meaning that there are identical rows in the input table. Combining with the two red rows, we may infer that a *deduplicate* transformation is performed to remove the two rows. Besides, two brown markers appear on the top side, which indicates the `id` between input and output tables have superset relation. A descending symbol is placed close to `total`, which indicates that the `total` column is sorted in descending order. By extending TACO, we can observe detailed relations between two tables. An exciting research direction is exploring visual encodings or novel designs to reveal richer characteristics of table changes effectively and intuitively.

7 DISCUSSION

In this section, we first discuss the limitations of our pipeline and the experiments. Then we present potential usage for future exploration.

7.1 Limitations

Evaluation. The evaluation of COMANTICS is limited in three aspects. To begin with, the quantitative experiments show that our pipeline could infer the transformation type with high Top-N accuracy. However, we did not assess the quality of the inferred parameters, which are critical in understanding semantics as well. In addition, we have not tried models other than the Siamese Convolutional Neural Network in Model-Based Inference. Lastly, the design space of table changes has not been explicitly evaluated in terms of descriptive, evaluative, and generative power. In future research, we plan to design comprehensive experiments to assess the quality of COMANTICS and the design space, and investigate how COMANTICS performs with different multi-class classification models.

Generalizability. Generalization across Python and R is difficult, and the current prototype only achieves 53.8% accuracy for Python/training and R/test, and 44.5% for R/training and Python/test. When performing the same transformation, function names and parameters
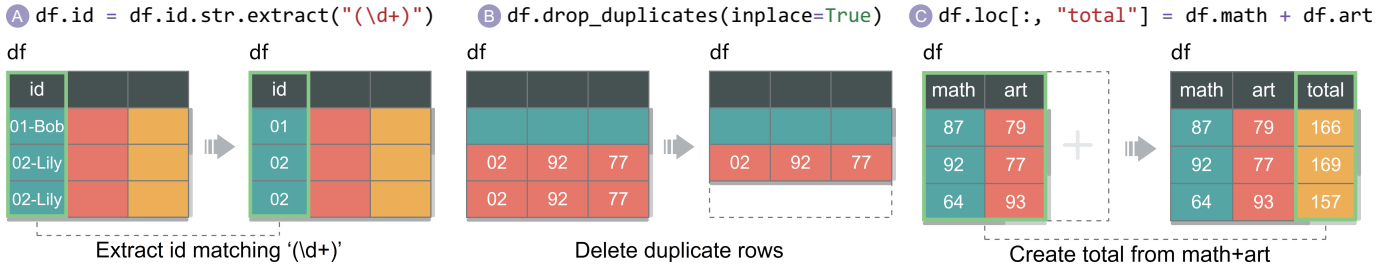
Fig. 6. Three examples that SOMNUS can support with the help of COMANTICS. Before integrating COMANTICS, SOMNUS does not support (a) the `extract` function, (b) non-assignment statements, and (c) non-functional assignments.
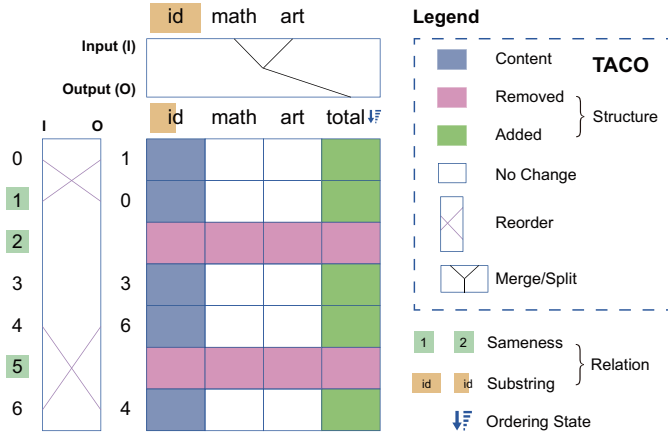

Fig. 7. A diff heatmap visualization of table differences generated by a modified version of TACO based on the characteristics of table changes detected from COMANTICS. The visual encodings enclosed by the dashed box in the legend are from TACO.

in different languages may vary while characteristics between tables are the same. We anticipate that the key factor to improve the generalizability is to align the semantics of function names and parameters among different languages. In addition, a large number of training and test samples are also critical for models to parse the semantics.

Scalability. Compared to prior work [50, 69], COMANTICS scales well in terms of the number of functions and parameters. However, our pipeline does not scale well in the number of transformation types. The mapping between transformation types and characteristics (described in Sec. 3.3) and rules for parameter inference (see Sec. 4.3) are crafted manually for each transformation. Our pipeline currently supports 30 types of transformations that are derived from the collected dataset. However, the transformation space has not yet been fully explored. Crafting rules for the entire space could be tedious and error-prone. In future iterations, we plan to investigate automatic approaches to build rules for the pipeline.

Capability. The limitations of COMANTICS in capability are threefold. First, COMANTICS infers one transformation with its parameters for a line of wrangling code. It is unable to reveal the semantics of a line of code containing multiple transformations. Second, it is not applicable to scripts with some imperative programming constructs, such as conditional statements and loops, which are commonly used in data wrangling. Third, it does not support function chaining due to the difficulty of obtaining intermediate data tables. Temporarily, this issue could be addressed by automatically converting the chained functions into individual lines. The three capability issues are critical to production usage, and we plan to address them in our future work.

7.2 Opportunities for New Usage

With the ability to infer the semantics of wrangling scripts, COMANTICS can introduce opportunities for practical applications in various domains. Apart from the three example applications described in

Sec. 6, we anticipate that our work can be integrated into productivity spreadsheet software and interactive data wrangling systems, including Microsoft Excel [45] and OpenRefine [23], to automatically record the history of table operation. This is helpful for documenting and sharing the transformation process and the table changes, and the history can be presented to augment the "undo" operation and used for auditing transformations [28]. In such a scenario, wrangling code is not explicitly generated. To adapt to these systems, feature engineering for function names and parameters should be transformed to the understanding of operation history in the interface.

Another possible usage of COMANTICS is to support a source-to-source compiler that converts code from one programming language to another [54]. Numerous source-to-source compilers have been proposed [54, 59, 63], which usually require large amounts of samples for training and testing. These techniques, however, can hardly support code translation in the domain of data wrangling for two reasons. First, samples for data wrangling are usually not well-documented, and manual annotation is laborious and time-consuming. Hence, learning-based approaches may result in poor performance without enough training samples. Second, subtle differences in the wrangling code may lead to different transformations. Learning the mapping among numerous functions and parameters across different languages is infeasible. COMANTICS narrows the gulf by aligning wrangling code with semantics. To be specific, we first infer the semantics for individual wrangling functions and their parameters using COMANTICS. Then, we establish bi-directional mappings between functions/parameters and semantics for different programming languages. Finally, the semantics act as bonds connecting functions in two languages.

8 CONCLUSION AND FUTURE WORK

In this paper, we have presented COMANTICS, a three-step pipeline that infers semantics for wrangling scripts. COMANTICS takes a piece of wrangling script and data tables as input, and outputs the semantics of each line of code consisting of the type of transformation and its parameters. Based on the observation that differences between input and output tables highly relate to the transformation type. We summarize a design space presenting characteristics of table changes, which further guides the design of COMANTICS. Experiments suggest that COMANTICS performs well in type inference. Further, three applications in different domains indicate that COMANTICS has good flexibility.

In the future, we plan to explore four promising research directions. First, we plan to improve the performance of COMANTICS by refining the Data Preprocessing step and Type Inference module. Second, we want to enhance its capability to meet the need for production usage. Third, we would like to incorporate COMANTICS in a broad range of applications, including spreadsheet software, data wrangling tool, and source-to-source compiler in the domain of data wrangling. Fourth, we plan to further evaluate COMANTICS with comprehensive experiments.

REFERENCES

[1] AQT - Advanced query tool - database query tool for DB2, Oracle, Sybase, SQL Server, MySQL. https://querytool.com/. Accessed: Feb 12, 2022.

[2] daff - data diffs in javascript, ruby, python, php, ... http://paulfitz.github.io/daff/. Accessed: Feb 12, 2022.

[3] DiffKit. http://www.diffkit.org/. Accessed: Feb 12, 2022.

[4] ExcelCompare. https://github.com/na-ka-na/ExcelCompare/. Accessed: Feb 12, 2022.

[5] Ridom SeqSphere+. https://www.ridom.de/u/Comparison_Table.html. Accessed: Feb 12, 2022.

[6] Variable Inspector. https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions/varInspector. Accessed: Feb 20, 2022.

[7] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. DataXFormer: A robust transformation discovery system. In *Proceedings of IEEE International Conference on Data Engineering*, pp. 1134–1145, 2016.

[8] A. Aditya and B. Akshit. Siamese Network Tensorflow. https://github.com/ardiya/siamesenetwork-tensorflow, 2019.

[9] L. Bartram, M. Correll, and M. Tory. Untidy Data: The unreasonable effectiveness of tables. *IEEE Transactions on Visualization and Computer Graphics*, 28(01):686–696, 2022.

[10] A. Bigelow, C. Nobre, M. Meyer, and A. Lex. Origraph: Interactive network wrangling. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology*, pp. 81–92, 2019.

[11] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information, 2017.

[12] R. Chen, D. Weng, Y. Huang, X. Shu, J. Zhou, G. Sun, and Y. Wu. Rigel: Transforming tabular data by declarative mapping. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[13] T. Dasu and T. Johnson. *Exploratory data mining and data cleaning*, vol. 479. John Wiley & Sons, 2003.

[14] L. E. R. Domingues, J. R. Ponciano, L. G. Nonato, and J. Poco. LegalVis: Exploring and inferring precedent citations in legal documents. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[15] 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 ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1–12, 2020.

[16] Y. Feng, R. Martins, J. Van Geffen, I. Dillig, and S. Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

[17] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 65–74, 2011.

[18] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1735–1742, 2006.

[19] Y. He, K. Ganjam, K. Lee, Y. Wang, V. Narasayya, S. Chaudhuri, X. Chu, and Y. Zheng. Transform-data-by-example (TDE) extensible data transformation in excel. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1785–1788, 2018.

[20] J. Heer and A. Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. *Information Visualization*, 13(2):111–133, 2014.

[21] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin. Deep code comment generation. In *Proceedings of IEEE/ACM International Conference on Program Comprehension*, pp. 200–210, 2018.

[22] Y. T. Hu, M. Burch, and H. v. d. Wetering. Visualizing dynamic data with heat triangles. *Journal of Visualization*, 25(1):15–29, 2022.

[23] D. Huynh. OpenRefine. https://openrefine.org. Accessed: Jan 29, 2022.

[24] J. P. Inala and R. Singh. WebRelate: integrating web data with spreadsheets using examples. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–28, 2017.

[25] D. Integration. Compare: A powerful data comparison tool. https://dispatchintegration.com/data-integration-software/compare/. Accessed: Feb 12, 2022.

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

[27] K. Kagkelidis, I. Dimitriadis, and A. Vakali. Lumina: an adaptive, automated and extensible prototype for exploring, enriching and visualizing data. *Journal of Visualization*, 24(3):631–655, 2021.

[28] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. Van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, 2011.

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

[30] S. Kasica, C. Berret, and T. Munzner. Table Scraps: An actionable framework for multi-table data wrangling from an artifact study of computational journalism. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):957–966, 2021.

[31] M. Khan, L. Xu, A. Nandi, and J. M. Hellerstein. Data tweening: incremental visualization of data transforms. *Proceedings of the VLDB Endowment*, 10(6):661–672, 2017.

[32] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[33] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proceedings of International Conference on Machine Learning*, 2015.

[34] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. BoostClean: Automated error detection and repair for machine learning, 2017.

[35] 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 ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1–12, 2021.

[36] C. Lewis and G. M. Olson. Can principles of cognition lower the barriers to programming? In *Proceedings of Empirical studies of programmers: second workshop*, pp. 248–263, 1987.

[37] B. Li, M. Yan, X. Xia, X. Hu, G. Li, and D. Lo. DeepCommenter: A deep code comment generation tool with hybrid lexical and syntactical information. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1571–1575, 2020.

[38] G. Li, R. Li, Z. Wang, H. C. Liu, M. Lu, and G. Wang. HiTailor: Interactive transformation and visualization for hierarchical tabular data. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[39] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *Proceedings of the International Conference on Intelligent User Interfaces*, pp. 97–106, 2009.

[40] J. Liu, N. Boukhelifa, and J. R. Eagan. Understanding the role of alternatives in data analysis practices. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):66–76, 2020.

[41] S. Liu, D. Peng, H. Zhu, X. Wen, X. Zhang, Z. Zhou, and M. Zhu. MulUBA: multi-level visual analytics of user behaviors for improving online shopping advertising. *Journal of Visualization*, 24(6):1287–1301, 2021.

[42] Z. Liu, S. B. Navathe, and J. T. Stasko. Network-based visual analysis of tabular data. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology*, pp. 41–50, 2011.

[43] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226, 2021.

[44] H. Mei, H. Guan, C. Xin, X. Wen, and W. Chen. DataV: Data Visualization on large high-resolution displays. *Visual Informatics*, 4(3):12–23, 2020.

[45] Microsoft. Microsoft Excel. https://office.microsoft.com/en-us/excel. Accessed: Feb 12, 2022.

[46] J. Morcos, Z. Abedjan, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. DataXFormer: An interactive data transformation tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 883–888, 2015.

[47] C. Niederer, H. Stitz, R. Hourieh, F. Grassinger, W. Aigner, and M. Streit. TACO: visualizing changes in tables over time. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):677–686, 2018.

[48] J. F. Pane. *A programming system for children that is designed for usability*. Carnegie Mellon University, 2002.

[49] F. Perez and B. E. Granger. Project Jupyter: Computational narratives as the engine of collaborative data science. *Retrieved September*, 11(207):108, 2015.

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

[51] Y. Qian and J. Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education*, 18(1):1–24, 2017.

[52] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras. *Principles of data wrangling: Practical techniques for data preparation.* " O'Reilly Media, Inc.", 2017.

[53] J. Reback, jbrockmendel, W. McKinney, J. V. den Bossche, T. Augspurger, P. Cloud, S. Hawkins, M. Roeschke, gfyoung, Sinhrks, A. Klein, P. Hoefler, T. Petersen, J. Tratner, C. She, W. Ayd, S. Naveh, M. Garcia, J. Darbyshire, J. Schendel, R. Shadrach, A. Hayden, D. Saxton, M. E. Gorelli, F. Li, M. Zeitlin, V. Jancauskas, A. McMaster, P. Battiston, and S. Seabold. pandas-dev/pandas: Pandas 1.4.0, Jan 2022.

[54] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample. Unsupervised translation of programming languages. In *Advances in Neural Information Processing Systems*, pp. 20601–20611, 2020.

[55] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt. Evaluating machine accuracy on ImageNet. In *Proceedings of International Conference on Machine Learning*, pp. 8634–8644, 2020.

[56] H. Shi. A Sequence-to-sequence Approach for Numerical Slot-filling Dialog Systems. In *Proceedings of the Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 272–277, 2020.

[57] N. Shrestha, T. Barik, and C. Parnin. Unravel: A fluent code explorer for data wrangling. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 198–207, 2021.

[58] T. Software. Tableau prep builder. `https://www.tableau.com/products/prep`. Accessed: Jan 29, 2022.

[59] T. S. Solutions. The Most Accurate and Reliable Source Code Converters. `https://www.tangiblesoftwaresolutions.com/`. Accessed: Feb 12, 2022.

[60] J. Sorva, V. Karavirta, and L. Malmi. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, 13(4):1–64, 2013.

[61] C. Sutton, T. Hobson, J. Geddes, and R. Caruana. Data diff: Interpretable, executable summaries of changes in distributions for data wrangling. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2279–2288, 2018.

[62] C. Tominski, G. Andrienko, N. Andrienko, S. Bleisch, S. I. Fabrikant, E. Mayr, S. Miksch, M. Pohl, and A. Skupin. Toward flexible visual analytics augmented through smooth display transitions. *Visual Informatics*, 5(3):28–38, 2021.

[63] transpiler. Universal-transpiler. `https://jarble.github.io/transpiler/`. Accessed: Feb 12, 2022.

[64] Trifacta. Data Wrangling Software and Tools - Trifacta. `https://www.trifacta.com/`. Accessed: Jan 29, 2022.

[65] Y. Wang, M. Yu, G. Shan, H.-W. Shen, and Z. Lu. VISPubComPAS: a comparative analytical system for visualization publication data. *Journal of Visualization*, 22(5):941–953, 2019.

[66] H. Wickham. *tidyr: Tidy Messy Data*, 2020. R package version 1.1.2.

[67] H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019.

[68] H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2021. R package version 1.0.4.

[69] K. Xiong, S. Fu, G. Ding, Z. Luo, R. Yu, W. Chen, H. Bao, and Y. Wu. Visualizing the scripts of data wrangling with SOMNUS. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[70] C. Yang, S. Zhou, J. L. Guo, and C. Kästner. Subtle bugs everywhere: Generating documentation for data wrangling code. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 304–316, 2021.