AI assistants: A framework for semi-automated, accountable, tooling-rich data wrangling

AIDA Project Team, The Alan Turing Institute

This document is submitted as a deliverable GDI 1.8: "A technical report on the use of digital assistants to support analyst workflows."

1 Introduction

According to Dasu and Johnson [4], 80% of data analysis is spent on *data wrangling* – an activity that encompasses obtaining data from a variety of semi-structured sources, making sense of data encoding, joining multiple datasets with mismatched structure, correcting erroneous records and filling missing data. Despite the recent rise of artificial intelligence and machine learning, data wrangling has, so far, largely eluded automation and remains a tedious and time-consuming manual task.

Data wrangling is hard to automate for many reasons. First, data is often big and diverse and hide hard-to-find special cases that can be only handled with human insight. Second, real-world data is heterogeneous. Automatic methods developed for one kind of data will often fail when more data is added. Third, data cleaning often requires tuning of many parameters and you often have to look at the results manually to see what worked. Finally, an error during data wrangling can have a disasterous effect on a project. An automatic tool can easily confuse interesting outliers for uninteresting noise in cases where a quick human glance would immediately spot the distinction. Data wrangling is often laborious because doing the work in a tedious manual way makes it possible to incorportate, every now and then, a simple but crucial human insight.

For these reasons, we argue that a major advance in data wrangling is best achieved by developing semi-automatic methods that allow us to effectively combine automation and scalability of machine learning methods with human insight. To capture the human-computer interaction pattern that is required for data wrangling, we introduce the concept of an *AI assistant*. An AI assistant is a component that interacts with a data analyst to guide them through a single data extraction, collection, integration, preparation or exploration task. AI assistants are semi-automated, accountable and tooling-rich:

1. **Semi-automated.** After being invoked on input data, the AI assistant runs an automated data wrangling algorithm (utilizing AI, machine learning or statistical analysis) and offers the analyst a best solution together with a list of suggestions to guide the process. The analyst can accept the results or choose one of the suggestions to inform the assistant. The loop is repeated until the AI assistant has no further suggestions or until the analyst accepts one of the result.

2. **Accountable.** Rather than opaquely turning *input data* into *cleaned data*, each interaction with the AI assistant results in a short data wrangling script in a language defined by each assistant. Representing the result as a script ensures reproducibility and increases transparency. In many cases, the script is human readable and allows the user to review what cleaning operations have been recommended by the AI assistant.

3. **Tooling-rich.** Finally, an AI assistant needs to play well with modern data science tooling. This means that offering the suggestions to the data analyst should be done through an interactive programming mechanism such as auto-complete. When reviewing suggestions, the data analyst should also immediately see a preview of running the script on the actual input data as, for example, when running code in Jupyter notebooks.

Several tools that satisfy the above definition of an AI assistant have been built in the past and we add a number of further concrete examples of AI assistants in this paper. However, the main contribution of this paper is that we describe the general structure of an AI assistant abstractly. We then use this structure to develop a optimization-based machine learning framework for definining AI assistants, but also to support AI assistants in a notebook-like data science environment.

A data analyst will need to interact with many assistants over the course of an analysis, each of which specializes in a single task. Having a unified abstract definition of an AI assistant provides a common framework, both theoretical and implementation, that authors of AI assistants and authors of data science tools can use in order to build a flourishing ecosystem for semi-automated, accountable, tooling-rich data wrangling.

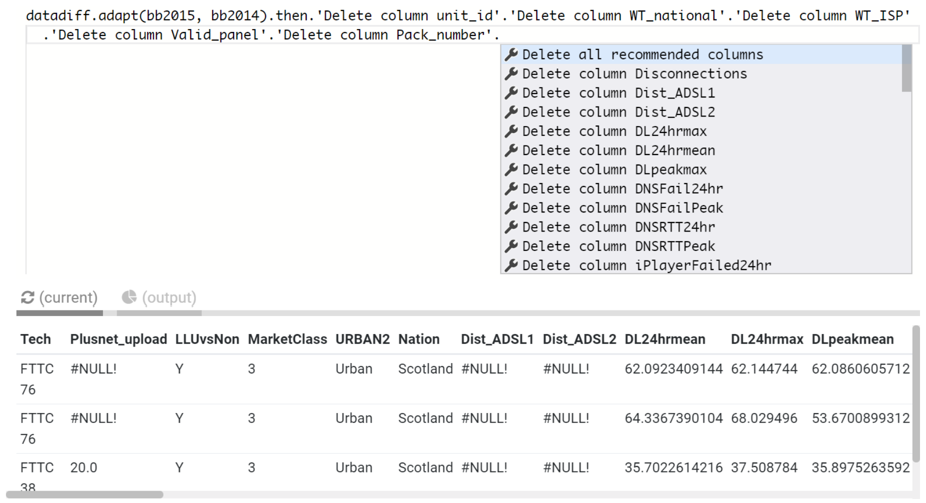
This paper explains what an AI assistant is, discusses AI assistants from the programming language perspective, as well as the machine learning perspective and gives a number of concrete examples. Our main contributions are:

- We give a formal definition of AI assistants (Section 3). The definition captures an interaction pattern where the user provides input to the AI assistant and the assistant finds the best solution given the input data and feedback from the user.

- We use our definition in two ways. First, we follow a machine learning perspective to give a framework for defining AI assistants (Section 3.3). Second, we follow the programming language perspective to discuss how to integrate AI assistants in data science tools (Section 3.4).

- We give a number of examples of AI assistants that all share the common structure, including assistants for reconciling multiple data files (Section 4.1.1), semi-automated parsing of CSV files (Section 4.2.1), adding semantic information to data (Section 4.2.3), inferring types of data (Section 4.3.2), and filling missing data (Section 4.3.3).

- We describe a concrete implementation of a notebook system for data wrangling that supports AI assistants (Section 5). Our system uses AI assistants and simplifies the data wrangling process by letting users easily control the results based on their human insights, giving rapid feedback and providing the ability to review and check resulting scripts.



```
datadiff.adapt(bb2015, bb2014).then.'Delete column unit_id'.'Delete column WT_national'.'Delete column WT_ISP'
  .'Delete column Valid_panel'.'Delete column Pack_number'.
```

```
Delete all recommended columns
Delete column Disconnections
Delete column Dist_ADSL1
Delete column Dist_ADSL2
Delete column DL24hrmax
Delete column DL24hrmean
Delete column DLpeakmax
Delete column DNSFail24hr
Delete column DNSFailPeak
Delete column DNSRTT24hr
Delete column DNSRTTPeak
Delete column iPlayerFailed24hr
```

Tech	Plusnet_upload	LLUvsNon	MarketClass	URBAN2	Nation	Dist_ADSL1	Dist_ADSL2	DL24hrmean	DL24hrmax	DLpeakmean
FTTC 76	#NULL!	Y	3	Urban	Scotland	#NULL!	#NULL!	62.0923409144	62.144744	62.0860605712
FTTC 76	#NULL!	Y	3	Urban	Scotland	#NULL!	#NULL!	64.3367390104	68.029496	53.6700899312
FTTC 38	20.0	Y	3	Urban	Scotland	#NULL!	#NULL!	35.7022614216	37.508784	35.8975263592

Figure 1: The datadiff AI assistant running inside Wrattler, a notebook-like environment for data science. The completion list shows data cleaning operations offered by an AI assistant and the preview shows the result of running the provided cleaning script.

2 AI assistants for data wrangling

A typical way of designing a tool to assist with a particular data wrangling challenge is to implement a set of functions (in R or Python) that take one or more input datasets and produce one or more datasets that do not suffer from the problem that the tool is solving. The user can call these functions on their dirty datasets and produce clean datasets. Such functions typically take some parameters that specify how cleaning should be done.

This simple approach has two problems. First, real-world data often contain numerous corner cases that require human insight. Getting the right results often involves laborious fiddling with function parameters. Second, we want the analysis to be accountable. The analyst should be able to understand what data transformations are performed. A function does not address these. It does not encourage human interaction and it is not accountable – it produces a result without indicating how and why.

In the following section, we introduce AI assistants informally through an example. We then contrast AI assistants with traditional data wrangling tools constructed as libraries of functions. A formal definition of an AI assistant follows in Section 3.

2.1 Introducing AI assistants

As an example, we consider the datadiff AI assistant. Datadiff takes two datasets that represent the same information, but are stored in an inconsisent format. For example, we might have measurements for two distinct years in two CSV files, stored in a mismatched format where the order of columns and ordering differs between them. As the column names also differ, this cannot be trivially fixed. Datadiff attempts to find corresponding columns by doing statistical analysis of the data in each column. It then generates a list of patches to reconcile the structure of the two datasets. A patch describes a data transformation that should be applied on the data, such as reorder certain columns or modify encoding in a categorical column according to a given mapping.

Datadiff has previously been implemented as an R function that takes two datasets and various parameters to adjust the likelihood of different types of patches. It then produces a list of patches. We developed an interactive version in the form of an AI assistant. The interactive version, running in our notebook environment for data science, is shown in Figure 1. We discuss datadiff in detail in Section 4.1.1 and the data science environment in Section 5.

In the example shown in Figure 1, the user has previously obtained two datasets, `bb2015` and `bb2014` that contain information about broadband speeds for years 2015 and 2014. The data analyst invokes the datadiff AI assistant in order to turn the format of 2015 data into the format used by the 2014 dataset. To do this, the user types `datadiff.adapt(bb2015, bb2014)`, which invokes the AI assistant. The user then types "." to get a data cleaning recommendation.

The AI assistant analyses the data and determines a list of patches that need to be applied in order to reformat the 2015 dataset. It then offers those patches in the auto-completion list, sorted from the most likely patch to the least likely patch. The user can proceed by selecting the first patch repeatedly (as in the above screenshot). After selecting a patch, the source code in the notebook changes and the preview is updated, showing the result of running the current cleaning script. This interactivity makes it easy to see possible data cleaning errors. If applying a patch does not lead to an expected result, the user can easily go back and select the second most likely patch.

2.2 Why are assistants more than functions

There are two key differences between AI assistants and data wrangling tools implemented as libraries of functions. First, the AI assistant is interactive. Using an AI assistant is a process where the user interacts with the assistant. At each step, the assistant performs its task and provides the user with the best possible solution with respect to the past interactions. It also provides the user with several further options to choose from. Those can constrain the solutions, resolve ambiguous corner cases or capture next steps that the AI assistant can follow. In this paper, the options are a finite list (where the user has to choose one), but they could also be open-ended (where the user can specify some input). The list of options is also sorted by ranking. If we choose the most likely option, the assistant can be seen as an automatic function.

Second, AI assistants do not directly transform data, but instead, generate scripts (or, more formally, expressions) in simple languages (domain-specific languages, or DSLs) that define how data should be transformed. The AI assistant can still use sophisticated machine learning algorithms (that do not produce an explanation of how they work) behind the scenes, but the end result will always be a script that can be executed reproducibly and often also reviewed and understood by the user. Note that each AI assistant can use its own domain-specific language – the language should be as small as possible, i.e. it should be able to express the data transformations that the AI assistant can recommend, but nothing more.

2.3 AI assistants in a notebook environment

The fact that AI assistants are more than just functions also means that we can build a number of interesting tools to support the data analysts interaction with an AI assistant. We discuss this aspect in detail in Section 5, but give a brief overview here.

An AI assistant is defined by three operations. The first operation provides the best solution to the particular data wrangling problem that the assistant is addressing, given past interactions with the human analyst. The solution is a script or an expression. The second operation allows us to run this expression on the input data and obtain cleaned data. Finally, the third operation produces a list of possible options that the analyst can choose from, given the input data and past interactions.

This design makes it possible to build a number of tools for integrating AI assistants into a data science workflow. Figure 1 shows two interesting features of a data wrangling environment leveraging AI assistants that we built:

- The screenshot shows a preview of the resulting dataset during an interaction with the AI assistant. This is obtained by running the script provided by the AI assistant on the input data. However, if the input data is too large, we can also run the provided script only on a part of the data to give a quick example of how the operation behaves.
- The auto-completion list showing possible patches is a general mechanism that can be used with any AI assistant. At each step of the process, the AI assistant provides a list of choices, alternatives or constraints that the analyst can choose from. A notebook environment can expose those via a completion list to let the analyst easily explore the options.
- In the example, we see the script generated by the AI assistant in the form of code as a list of accepted operations. The script is written in a small language that is specific for each assistant, e.g. a list of patches such as "delete column". This script can be translated to any programming language such as R and Python, which means that one can use an AI assistant during data wrangling, but then end up with code that can be used in production in any programming language.

3 Formal model of AI assistants

In this section, we give a formal definition of AI assistants. An AI assistant in our model is an abstract algebraic structure (akin to a monoid or a group in algebra). The definition captures the common pattern that is shared by all the concrete AI assistants that we discuss later in Section 4.

By capturing the common pattern in an abstract algebraic way, we are also able to describe a number of uses and methods that can be shared by multiple AI assistant. We consider two such common uses. In Section 3.3, we describe the problem of finding the best cleaning script through a machine learning perspective and in Section 3.4, we use the common structure to describe how programming environment, such as the notebook system shown in Figure 1 uses an AI assistant.

3.1 The structure of an AI assistant

Although all AI assistants share a common structure, there is much that remains specific to each AI assistant. As mentioned earlier, the expressions or scripts (written e)[1] that an assistant produces differs for each assistant. In addition, each assistant also has its own specification of human interactions H and an assistant-specific notion of data $\mathcal{D}$. The notion of data is shared between *some* of our assistants, but not all. We consider it desirable to unify those across all AI assistants, but we believe tht this requires further research to ensure that our unified notion of data is sufficiently general.

Each AI assistant needs to provide three operations. First, given input data $\mathcal{D}$ and past human interactions H, it recommends the best expression (or a script) to solve the data wrangling problem. We write this as $best_{\mathcal{D}}(H)$. Second, human interacts with the AI assistant by selecting one from the options offered by the assistant (as illustrated in Figure 1). The second function provided by the AI assistant generates those choices. Given input data $\mathcal{D}$ and a trace of past human interactions H, the function $choice_{\mathcal{D}}(H)$ generates a list of options $H_1, \ldots, H_k$. To bootstrap this process, the AI assistant also needs to define initial (or empty) human interaction[2] trace H_0. Finally, the AI assistant also needs to define how an expression e is evaluated. Given input data $\mathcal{D}$ and an expression e, the function $f(e, \mathcal{D})$ returns a new transformed dataset according to e:

Definition 1 (AI assistant)**.** Given a sets of expressions e, possible datasets $\mathcal{D}$ and human intervention specifications H, an *AI assistant* is a tuple $(H_0, f, best, choices)$ where H_0 is a specification capturing no human interactions, f is a function and $best$ and $choices$ are families of functions indexed by input data:

- $f(e, \mathcal{D}) = \mathcal{D}'$
- $best_{\mathcal{D}}(H) = e$
- $choices_{\mathcal{D}}(H_0) = (H_1, H_2, \ldots, H_k)$

The function f takes an expression produced by the assistant together with an input dataset and returns a new dataset, processed according to the script. As noted earlier, the notion of dataset is specific to each AI assistant, but it is similar in all cases we consider – typically, one or multiple data tables (or data frames) with some meta-data that add type information or semantic information about the data.

The AI assistant then provides two functions $best_{\mathcal{D}}$ and $choices_{\mathcal{D}}$ that depend on input data $\mathcal{D}$. This can be the actual input dataset to be cleaned or a smaller subset representing training data. In the latter case, the user can then use the function f to run the obtained expression on the full dataset. The two functions also rely on a description of past human interactions with the assistant. The function $best_{\mathcal{D}}$ returns a single best data cleaning script given a trace of human interaction H. The function $choices_{\mathcal{D}}$ takes a current trace of past human interactions and returns an ordered list of choices that the analyst can make next. Note that this is an ordered list and so the AI assistant can offer the most likely options first.

[1]We will be using the term "expression" in this paper, which is a terminology used in the context of programming languages. Data scientists can think of e as a script or even just a set of parameters.

[2]Note that we use the notation H_0 for initial or empty *human interaction* rather than a null hypothesis in statistical testing. This paper does not talk about hypothesis testing, so this should not cause any ambiguity.

$$
\begin{aligned}
M &= \{\text{""}, \text{"NA"}, \text{"na"}, \text{"\#N/A"}, \text{"null"}, \text{"-1"}\} \\
\mathcal{D} &= \{v_{i,j}\} \\
e &= \{\mathsf{missing}(m_1), \ldots, \mathsf{missing}(m_k)\} \\
H &= \{\mathsf{valid}(m_1), \ldots, \mathsf{valid}(m_l)\}
\end{aligned}
$$

Figure 2: Definitions of data, expressions and human interactions for a minimal AI assistant that identifies and removes missing values

3.2 Minimal assistant for detecting missing values

To illustrate our definition, we now give a minimal example of an AI assistant that lets the user interactively remove various kinds of missing values. The input data for the assistant is a table with values as strings, including a special value $\perp$ that denotes a missing value. However, the input can also contain values such as "na" or "#N/A" which may or may not indicate missing values. The assistant lets the user interactively choose which of special (pre-defined) strings should be treated as missing values. Evaluating the constructed expression (cleaning script) then repalces all such missing values with the special value $\perp$.

Figure 2 defines data, expressions and human interactions of the assistant. M is a pre-defined set of strings that might indicate missing values; $\mathcal{D}$ is a matrix of values $v_{i,j}$, where each value is either a string or a missing value $\perp$. An expression e is a set of operations written as $\mathsf{missing}(m)$, which indicate that a value m should be trated as missing. For example, an expression $\{\mathsf{missing}(\text{"na"}), \mathsf{missing}(\text{"\#N/A"})\}$ indicates that all "na" and "#N/A" values should be replaced by $\perp$.

Human interactions specify a set of constraints which indicate that certain strings which might indicate missing value are actually valid. When interacting with the assistant, we start with an empty set of constraints $H_0 = \emptyset$. The assistant then treats all values in the set M as missing values that should be replaced with $\perp$. However, it also offers the analyst an option to add the valid constraint for all the values in M and so the analyst can, for example, add $\mathsf{valid}(\text{"-1"})$ if they decide that -1 is a valid value.

The operations of the sample AI assistant are defined in Figure 3. The evaluation function f returns a new dataset where each value $v_{i,j}$ is replaced with $\perp$ if $\mathsf{missing}(v_{i,j})$ is a member of the set representing the cleaning expression. The $best_{\mathcal{D}}$ operation produces an expression indicating that values from M should be treated as missing as long as they appear somewhere in the input data and were not explicitly marked as valid by the analyst, i.e. are not included as valid among constraints H. Finally $choices_{\mathcal{D}}$ allows the user to add valid constraints one by one – given an existing set of constraints H produced by earlier human interaction, it offers new sets where one of the remaining values from M that is not already marked as valid is added to the constraint set.

The assistant presented in this section is not very practically useful and it does not use any AI technique to make good recommendations, but it illustrates how AI assistants work and how they can be defined. In our actual examples, discussed in Section 4, the $best$ and $choices$ operations are more sophisticated.

$$\begin{aligned}
&f(e, \{v_{i,j}\}) = \{v'_{i,j}\} \\
&\qquad \text{where } v'_{i_j} = \begin{cases} \bot & (\text{when } \mathsf{missing}(v_{i,j}) \in e) \\ v_{i,j} & (\text{otherwise}) \end{cases}
\end{aligned}$$

$$\begin{aligned}
&best_{\{v_{i,j}\}}(H) = \{\mathsf{missing}(m) \mid m \in M, \forall i \in 1 \ldots l.\ m \neq m_i, \exists i,j.\ m = v_{i,j}\} \\
&\qquad \text{where} \quad H = \{\mathsf{valid}(m_1), \ldots, \mathsf{valid}(m_l)\}
\end{aligned}$$

$$\begin{aligned}
&choices_{\mathcal{D}}(H) = (H \cup \{\mathsf{valid}(m_1)\}, \ldots, H \cup \{\mathsf{valid}(m_k)\}) \\
&\qquad \text{where} \quad H = \{\mathsf{valid}(m_1), \ldots, \mathsf{valid}(m_l)\} \\
&\qquad\qquad \{m_1, \ldots, m_k\} = \{m \mid m \in M, \forall i \in 1 \ldots l.\ m \neq m_i\}
\end{aligned}$$

Figure 3: Definitions of f, $best$ and $choices$ operations for a minimal AI assistant that identifies and removes missing values

3.3 AI assistants as optimization problem

Although each AI assistant could be defined independently as in the previous example, the common structure also allows us to capture some commonalities among many of the assistants we develop. In particular, we can define an optimization framework that captures the general operation of any AI assistant. For specific AI assistants this optimization problem can be further simplified, for instance as a problem with constraints or a Bayesian inference problem. In this section, we describe a model that is shared by the assistant for parsing CSV files (Section 4.2.1), the assistant for inferring types of columns in a data table (Section 4.3.2) and an assistant for filling missing data (Section 4.3.3).

As before, let e denote an expression that defines a data wrangling operation and let H denote the human interaction trace. Human interaction can take the form of a constraint set, as in the missing value assistant described above, or could for instance define a probability distribution in a Bayesian inference problem. We further introduce the set E_H as the set of all allowed expressions given the human interaction H. We then define the operation $best_{\mathcal{D}}$ for all AI assistants through the optimization problem

$$best_{\mathcal{D}} = \arg\max_{e \in E_H} Q_H(\mathcal{D}, e)$$

where $Q_H(\mathcal{D}, e)$ denotes an objective function that assigns a numerical score to an expression e applied to the data $\mathcal{D}$ taking into account the human interaction H. Note that H occurs in both the set of allowed expressions E_H and the parameterization of the objective function Q_H. This is because, in general, an AI assistant could allow human interaction to affect *both* aspects of the optimization problem. For instance, human interaction could limit the allowed expressions or could for instance affect hyperparameters of an objective function. The definitions of the operations f and $choices_{\mathcal{D}}$ are specific to each AI assistant and we discuss them separately in Section 4.

Detecting missing values. As an example, consider the AI minimal assistant for detecting missing values presented in the previous section. In this case, the human interaction restricts the set of possible expressions E_H to those that do not label as $\mathsf{missing}$ any of the strings that were explicitly marked by the user as valid in the human interaction H. The optimization then finds an expression that maximizes the number of values that the expression e marks as missing and that appear in the data $\mathcal{D}$. More formally:

$$\begin{aligned}
&best_{\mathcal{D}}(H) = \arg\max_{e \in E_H} Q(\mathcal{D}, e) \\
&\quad \text{where} \quad Q(\mathcal{D}, e) = \textstyle\sum_{x \in e} I[\exists v \in \mathcal{D} : \mathsf{missing}(v) = x] \\
&\qquad\qquad\quad E_H = \{e : \mathsf{missing}(m) \notin e,\ \forall m . \mathsf{valid}(m) \in H\} \\
&\qquad\qquad\quad I[p] = 1 \text{ (when } p\text{)} \quad 0 \text{ (otherwise)}
\end{aligned}$$

Here, $I[\cdot]$ is an indicator function that returns 1 if its argument is true and 0 otherwise; H is defined as the set of valid constraints for all strings that should *not* be considered missing values, and E_H is a set of all expressions that are allowed given a human interaction trace H. The objective function Q counts the number of values marked as $\mathsf{missing}$ in e that also appear somewhere in the data $\mathcal{D}$. Note that for this specific example $Q_H(\mathcal{D}, e) = Q(\mathcal{D}, e)$ as the human interaction only affects the set of allowed expressions E_H.

3.4 AI assistants as programming tools

The formal definition of an AI assistant presented in Section 3.1 provides a common interface between a wide range of data wrangling tools and a programming environment. This means that we can:

i. Develop programming tools for interactive data wrangling without knowing the specific details of individual AI assistants – as long as we only use the common interface.

ii. Develop AI assistants without knowing the specific details of any programming system that will be using them.

In this section, we outline how to use the common AI assistant interface in a programming language that supports AI assistants through autocomplete as shown in Figure 1. Recall that, in the earlier example, the datadiff AI assistant offered the user a list of patches that the user can choose from. After typing "." the user sees a list of available patches. They can then choose one of the patches and type "." again to see the remaining patches. By typing "." and choosing options, the user is navigating through a tree, although the tree is lazily evaluated. The user typically does not try all the possible options and so most of the tree is never explored.

Using the definition of an AI assistant (Definition 1), we can now explain how the interactive process works more formally. After obtaining the input data $\mathcal{D}$ for the AI assistant, we obtain the initial (empty) trace of human interaction H_0 together with the three functions that define the AI assistant, i.e. $best_{\mathcal{D}}$, $choices_{\mathcal{D}}$ and f. In the concrete example discussed in Section 2.1, this was done by calling `datadiff.adapt` using `datadiff.adapt(bb2015, bb2014)`.

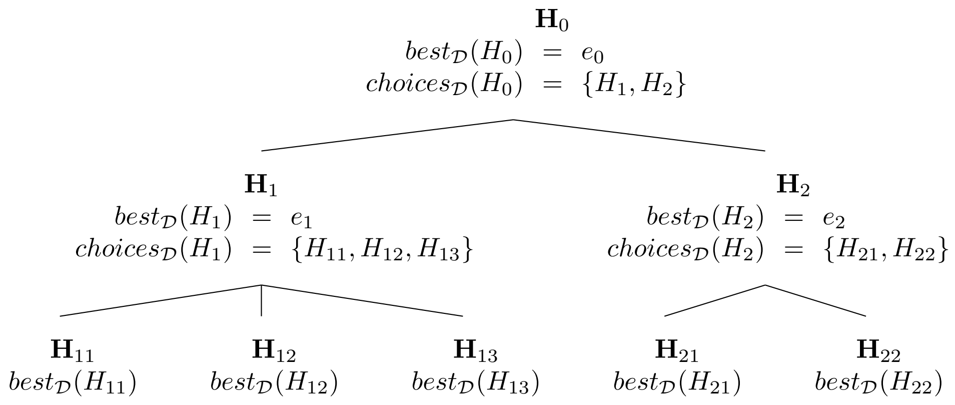

$\mathbf{H}_0$
$best_{\mathcal{D}}(H_0) = e_0$
$choices_{\mathcal{D}}(H_0) = \{H_1, H_2\}$
$\mathbf{H}_1$
$best_{\mathcal{D}}(H_1) = e_1$
$choices_{\mathcal{D}}(H_1) = \{H_{11}, H_{12}, H_{13}\}$
$\mathbf{H}_2$
$best_{\mathcal{D}}(H_2) = e_2$
$choices_{\mathcal{D}}(H_2) = \{H_{21}, H_{22}\}$
$\mathbf{H}_{11}$
$best_{\mathcal{D}}(H_{11})$
$\mathbf{H}_{12}$
$best_{\mathcal{D}}(H_{12})$
$\mathbf{H}_{13}$
$best_{\mathcal{D}}(H_{13})$
$\mathbf{H}_{21}$
$best_{\mathcal{D}}(H_{21})$
$\mathbf{H}_{22}$
$best_{\mathcal{D}}(H_{22})$

Figure 4: A tree construction based on an AI assistant that allows the user to navigate through possible solutions to a data wrangling problem.

We now need to construct a tree that the user can interactively explore, using just the operations provided by the common interface of an AI assistant. The construction is illustrated in Figure 4. Each node of the tree is identified by a trace of human interaction H. From this, we can obtain the expression (or data cleaning script) associated with the node by using $e = best_{\mathcal{D}}(H)$. The programming tool can then also use the expression e to give the user a preview of what the data, cleaned using the expression e, look like. This is done using $f(e, \mathcal{D})$.

When constructing the tree, we start with the empty human interaction trace H_0 and use it as the root node. We then use the $choices_{\mathcal{D}}$ operation to generate possible options that the user can choose from if they are not happy with the data wrangling expression obtained based on H_0. This is done by calling $choices_{\mathcal{D}}(H_0)$. In Figure 4, this returns just two options, H_1 and H_2. The user can then choose one of the branches to follow and they will then get the corresponding data wrangling expression for the particular brach together with data preview and further options to choose from. The process continues recursively until the user accepts the offered expression, or until the assistant offers no further choices, i.e. $choices_{\mathcal{D}}(H) = \emptyset$.

The most important aspect of the construction described in this section is that it does not depend on which particular AI assistant are we using. As long as the AI assistant provides the common interface captured by Definition 1, it will be possible to use it interactively through a programming tool that lets the user navigate through a tree of possible data wrangling expressions to find the most appropriate one. It is also worth noting that this is not the only possible programming tool that can be built based on the common interface – one might, for example, build a tool that searches the tree automatically to find the most appropriate cleaning expression based on some externally provided scoring function without any human interaction.

4 AI assistants for data wrangling

In this section, we present six AI assistants for data wrangling. All of the assistants are based on our recent work on tools for data wrangling, developed in the context of the AIDA (AI for Data Analytics) project. In this paper, we present interactive versions of such tools, which allow the human user to interactively guide the AI assistant.

We show a full range of possible AI assistants. In Section 4.1, we give one example that satisfies the definition of an AI assistant (Definition 1), but does not fit in the optimization framework introduced in Section 3.3. In Section 4.2, we show three examples that use the basic version of the optimization framework and, finally, Section 4.3 shows two examples that fit in richer, Bayesian version of our optimization framework.

4.1 AI assistants based on upfront optimization

We believe that the most useful kind of AI assistants will follow the optimization framework outlined in Section 3.3. In those cases, the $best$ operation performs optimization after it is given a trace of past human interaction. In other words, it aims to find the best cleaning expression, given a past human interaction according to some scoring function. However, our definition is more general.

One AI assistant where the $best$ operation does not perform optimization is the datadiff assistant shown earlier in Section 2.1. The assistant itself performs optimization to determine the best set of patches, but this happens only once, when the assistant is invoked for the first time. The $best$ operation then merely allows the user to select some of those pre-computed patches. We describe this assistant in the next section, but we later revisit the topic and discuss a better, optimization-based version in Section 4.2.2.

4.1.1 Simple datadiff assistant

The datadiff package [9] takes two tabular datasets that are in inconsistent formats, but represent two sets of observations about the same entity. An example is measurements of broadband speed in two different yers, stored as two tabular files with missing or additional columns, using different order of columns and different column names.

Datadiff expressions. Datadiff generates a list of patches that can be applied to one dataset in order to convert it to the format used by the second dataset. An expression e produced by the datadiff assistant is a list of patches p, which can be one of the following:

$$\begin{array}{rcll}
p & = & \mathsf{permute}(\pi) & \text{(Reorder columns using a given permutation)} \\
 & | & \mathsf{recode}(i, [s_1 \mapsto s_2, \ldots]) & \text{(Recode categorical values using in column } i\text{)} \\
 & | & \mathsf{linear}(i, a, b) & \text{(Apply linear transform } va + b \text{ to a column } i\text{)} \\
 & | & \mathsf{delete}(i) & \text{(Drop the column at index } i\text{)} \\
 & | & \mathsf{insert}(i, d) & \text{(Insert column at index } i \text{ with values } d\text{)} \\
e & = & p_1, \ldots, p_n & \text{(An expression is an ordered list of patches)}
\end{array}$$

The datadiff package, as described earlier, is a non-interactive R function that takes two datasets and produces a list of patches. It takes a couple of parameters that specify which patches should datadiff generate (together with weights to tweak the optimization).

$$
\begin{aligned}
& best_{D,D'}(H) = p_{i_1}, \ldots, p_{i_k} \\
& \qquad \text{where} \quad \begin{aligned} p_1, \ldots, p_n &= \mathit{ddiff}(D, D') \\ \{i_1, \ldots, i_k\} &= H \end{aligned}
\end{aligned}
$$

$$
\begin{aligned}
& choices_{D,D'}(H) = H \cup \{j_1\}, \ldots, H \cup \{j_k\} \\
& \qquad \text{where} \quad \begin{aligned} p_1, \ldots, p_n &= \mathit{ddiff}(D, D') \\ \{j_1, \ldots, j_k\} &= \{j \mid j \in 1 \ldots n, j \notin H\} \end{aligned}
\end{aligned}
$$

Figure 5: Operations of a pre-computation based datadiff AI assistant

Example of using datadiff. In this section, we describe a simple AI assistant that invokes the non-interactive R function and then allows the user to select and apply individual patches. The user can review each patch to make sure that it is a reasonable transformation. For example, consider the following two datasets:

```
Name, City
Joe, London
Jane, Edinburgh
Jim, London
```

```
City, Name, Count
Cardiff, Alice, 1
Cardiff, Bob, na
Edinburgh, Bill, 2
```

If we ask datadiff for a list of patches to transform the dataset on the right-hand side to the format of the dataset on the left-hand side, we might get the following patches (assuming columns are indexed from 1):

$$\mathsf{delete}(3);\ \mathsf{permute}(2,1);\ \mathsf{recode}(2, [\text{"Cardiff"} \mapsto \text{"London"}])$$

Datadiff correctly infers that we need to drop the newly added `Count` column and that the order of `Name` and `City` has been switched. It might also suggest that `City` is a categorical column where the encoding has been changed, which is not the case here (but this would be a reasonable guess if one dataset contained values "true" and "false" while the other contained "yes" and "no").

Datadiff AI assistant. The simple version of the datadiff AI assistant discussed in this section invokes the standard datadiff function and then allows the user to accept individual patches. We assume that the standard datadiff function takes two data tables, D and D' and outputs a list of patches:

$$\mathit{ddiff}(D, D') = p_1, \ldots, p_n$$

The input $\mathcal{D}$ for the AI assistant is thus a pair of data tables D, D'. Human interactions are simply sets of accepted patches, i.e. $H \subseteq \{1, \ldots n\}$. The *best* and *choices* operations of the AI assistant are shown in Figure 5. The *best* operation selects the patches returned by *ddiff* that have been accepted by the user and so their indices appear in H. The *choices* operation generates indices $j_1, \ldots, j_k$ of patches that have not yet been selected, and offers a list of possible human interactions that are formed by H together

$$
\begin{array}{l}
H = \{0\},\ e = \mathsf{delete}(\ldots) \\
\qquad \longrightarrow H = \{0, 1\},\ e = \mathsf{delete}(\ldots); \mathsf{permute}(\ldots) \\
\qquad\qquad \longrightarrow H = \{0, 1, 2\},\ e = \mathsf{delete}(\ldots); \mathsf{permute}(\ldots); \mathsf{recode}(\ldots) \\
\qquad \longrightarrow H = \{0, 2\},\ e = \mathsf{delete}(\ldots); \mathsf{recode}(\ldots) \\
\qquad\qquad \longrightarrow H = \{0, 1, 2\},\ e = \mathsf{delete}(\ldots); \mathsf{recode}(\ldots); \mathsf{permute}(\ldots) \\
H = \{1\},\ e = \mathsf{permute}(\ldots) \\
\qquad \longrightarrow H = \{1, 0\},\ e = \mathsf{permute}(\ldots); \mathsf{delete}(\ldots) \\
\qquad\qquad \longrightarrow H = \{1, 0, 2\},\ e = \mathsf{permute}(\ldots); \mathsf{delete}(\ldots); \mathsf{recode}(\ldots) \\
\qquad \longrightarrow H = \{1, 2\},\ e = \mathsf{permute}(\ldots); \mathsf{recode}(\ldots) \\
\qquad\qquad \longrightarrow H = \{1, 2, 0\},\ e = \mathsf{permute}(\ldots); \mathsf{recode}(\ldots); \mathsf{delete}(\ldots) \\
(\ldots)
\end{array}
$$

Figure 6: A tree generated by a basic datadiff AI assistant. The parameters of the patches are omitted – they are the same as in the full list of patches shown above.

with one of the remaining indices. The initial human interaction H_0 is an empty set. We omit the f operation in the definition, because this is the same as the function that applies patches in the standard datadiff R package.

Example of a constructed tree. Earlier, we considered an example where the fully automatic datadiff function generates three patches and the user only wants to accept the first two. This is done by navigating through a tree structure that is constructed as discussed in Section 3.4. We show the tree constructed in this particular case in Figure 6.

At the first level, the user can choose one of the three patches (we only show subtrees for the first two). The patches are sorted by their rating and so delete is offered before permute. In the second level, the user can choose one of the remaining patches. Assuming the user follows the first branch and chooses delete, they can now choose to apply permute or recode (the first option being more likely). In the last third level, the AI assistant offers the remaining patch. The user can stop before reaching the leaf and choose just $\mathsf{delete}(3);\ \mathsf{permute}(2, 1)$ as the result, ignoring the recode patch.

This example shows why AI assistants provide a more convenient developer interface. The recode patch would, in some cases, be a reasonable data cleaning recommendation, but a human analyst can look at the values and see that this is not the case here. (We could imagine an assistant that integrates semantic information into the process and recognizes that the values represent cities. This solves the specific problem shown here, but leaves many similar problems unsolved.) When using datadiff as a cleaning function in R, we would get all three patches, apply them and then (hopefully) notice the incorrect recoding. We could fix the result by tweaking the parameters of the function, e.g. by lowering the threshold for recode patches. When using an AI assistant, the analyst can review all patches instantly and skip the unsuitable ones.

As before, the data wrangling expressions can be evaluated using the *eval* function (to get previews when writing the code in a notebook environment), but they can also be translated to simple R or Python scripts for production use.

4.2 Optimization based AI assistants

Below we present three AI assistants that follow the general optimization framework presented in Section 3.3, but do not follow a Bayesian inference framework. The latter are presented in the next section.

4.2.1 CleverCSV assistant for CSV parsing

The goal of CleverCSV [11] is to extract a data table from a CSV file without any knowledge about the structure of the file (i.e. file encoding, CSV dialect, presence of headers, location of the table, etc.). The CleverCSV AI assistant defines a CSV parser that takes an expression and outputs a data table. The human intervention in the AI assistant is primarily in the form of *constraints* on the allowed values of the various parsing parameters.

The expression e in CleverCSV is a set of operations from the following language:

$$\begin{array}{rcll} e & = & \mathsf{encoding}(E) & \text{(Use } E \text{ as the file encoding)} \\ & | & \mathsf{delimiter}(d) & \text{(Fields are separated by character } d\text{)} \\ & | & \mathsf{quotechar}(q) & \text{(Quoted fields are surrounded by } q\text{)} \\ & | & \mathsf{escapechar}(c) & \text{(The escape character is } c\text{)} \\ & | & \mathsf{header}(h) & \text{(There is a header at row } h\text{)} \\ & | & \mathsf{skipinitial} & \text{(Skip first space after delimiter)} \\ & | & \mathsf{range}(n,m,j,k) & \text{(There is a table from row } n \text{ to } m \text{ and column } j \text{ to } k\text{)} \end{array}$$

Some of these operations (such as header and range) can be provided multiple times when there is more than one data table in the file. At the moment, CleverCSV detects the dialect of the CSV file, i.e. the delimiter d, quote character q, and escape character c, through optimization of an objective function. The remaining parsing parameters available in the language can be supplied through human interaction. Thus we define human interaction for CleverCSV by a tuple $H = (e', U, V, W)$ where e' is the part of the expression that contains the non-dialect parameters and U, V, and W are respectively the set of allowed delimiters, quote characters, and escape characters. Then,

$$\begin{aligned} E_H \;=\; & \{e' \cup \{\mathsf{delimiter}(d)\} \cup \{\mathsf{quotechar}(q)\} \\ & \cup \{\mathsf{escapechar}(c)\} : \forall d \in U, q \in V, c \in W\} \end{aligned}$$

CleverCSV detects the dialect parameters by optimising a data consistency measure $Q(\mathcal{D}, e)$ where $\mathcal{D}$ denotes the CSV file. This data consistency measure only depends on the part of the expression that defines the possible dialects (i.e. d, q, and c). The data consistency measure is given by a product of two functions, $P(\mathcal{D}, e)$ and $T(\mathcal{D}, e)$ that respectively denote a *pattern* score and a *type* score. Full definitions of these functions are given in the CleverCSV paper.

The initial human interaction, H_0, is constructed by detecting the encoding using existing methods, taking the largest range possible, and by constructing U, V, and W using the procedure described in the CleverCSV paper.

The function $f(e, \mathcal{D})$ in CleverCSV is a CSV parser that takes the expression and a CSV file as input and produces a parsing result consisting of one or more data tables. The function $choices_D(H)$ generates tuples for human interaction that shrink or expand the sets U, V, or W and allow the user to extend e'.

4.2.2 Optimization based datadiff assistant

The datadiff AI assistant discussed in Section 4.1.1 is not optimal – it performs a global optimization up-front and then merely allows the user to choose some of the patches. This is problematic, because the global optimization can be slow (so the initial step might take too much time) and the result of the optimization does not improve based on the user input. If the user could specify that a certain column should not be recoded, a better datadiff assistant could run an optimization taking this constraint into account and, say, match columns differently.

Human interactions. A better datadiff AI assistant generates expressions in the same domain-specific language as the one discussed earlier (list of patches p), but human interactions are defined differently. Rather than keeping indices of accepted patches, we now maintain a list of constraints. The assistant then generates the best possible patch given certain constraints, together with a choice of most likely constraints the user might want to add. Human interaction H is a set of constraints c defined as:

$$\begin{array}{rcll} c & = & \mathsf{norecode}(i) & \text{(Values in a categorical column } i \text{ should not be recoded)} \\ & | & \mathsf{notransform}(i) & \text{(Values in a numerical column } i \text{ should not be transformed)} \\ & | & \mathsf{nodelete}(i) & \text{(Column } i \text{ should not be deleted)} \\ & | & \mathsf{match}(i, j) & \text{(Column } i \text{ should be mapped to sample column } j\text{)} \end{array}$$

The first three constraints specify that certain patches should not be generated. The last one is more interesting as it allows the user to explicitly give a mapping for a column (but unless other constraints are given, datadiff can still attempt to transform or recode the column to get a better match). This is likely not a complete set of constraints, but it is sufficient to illustrate the idea.

Datadiff as an optimization problem. The optimization based datadiff AI assistant can now be formulated using our general optimization framework. The $best$ operation aims to find the most suitable set of patches that satisfy the constraints specified by the user:

$$best_{\mathcal{D}} = \arg\max_{e \in E_H} Q_H(\mathcal{D}, e)$$

In case of datadiff, the scoring function Q_H is just Q and does not depend on H. This is the same scoring function that the existing datadiff R package uses to evaluate the suitability of a set of patches. However, the AI assistant uses E_H to restrict which sets of patches are evaluated. Assuming E is a set of all possible lists of patches (for a given dataset), E_H contains all lists of patches from E that are valid with respect to constraints H:

$$E_H = \{p_1, \ldots, p_k \in E, \forall i \in 1 \ldots k.\ \mathsf{valid}_H(p_i)\}$$

The valid_H predicate is defined as follows:

$$\begin{aligned}
\mathsf{valid}_H(\mathsf{permute}(\pi)) &\;\; \textit{iff} \;\; \forall \mathsf{match}(i,j) \in H.\ (i,j) \in \pi \\
\mathsf{valid}_H(\mathsf{recode}(i,[\ldots])) &\;\; \textit{iff} \;\; \mathsf{norecode}(i) \notin H \\
\mathsf{valid}_H(\mathsf{linear}(i,a,b)) &\;\; \textit{iff} \;\; \mathsf{norecode}(i) \notin H \\
\mathsf{valid}_H(\mathsf{delete}(i)) &\;\; \textit{iff} \;\; \mathsf{nodelete}(i) \notin H \\
\mathsf{valid}_H(\mathsf{insert}(i,d)) &
\end{aligned}$$

Example of a constructed tree. The $choices$ operation of the optimization based datadiff AI assistant uses a set of heuristics to offer possible constraints. This does not follow a particular principled approach and so we explain it using an example tree that the assistant produces for the problem discussed earlier in Section 4.1.1.

We start with an empty set of constraints $H_0 = \{\}$ and use $best$ to obtain a data cleaning DSL expression e and potential sets of constraints $H_1, \ldots, H_k$. As before, we generate a sub-tree for each set of constraints by calling the $choose$ function recursively. In other words, following a path through the tree would add more and more constraints to the set H and each node will come with the best DSL expression e that datadiff can generate under the constraints determined by the path taken through the tree. The following illustrates (a small part of) the tree that the assistant generates:

$$\begin{aligned}
&H = H_0 = \emptyset,\ e = \mathsf{delete}(\ldots);\ \mathsf{permute}(\ldots);\ \mathsf{recode}(\ldots) \\
&\quad \longrightarrow H = [\mathsf{norecode}(2)], e = \mathsf{delete}(\ldots);\ \mathsf{permute}(\ldots); \\
&\quad\quad \longrightarrow H = [\mathsf{norecode}(2); \mathsf{nodelete}(3)], p = 0.2, e = (\ldots) \\
&\quad\quad \longrightarrow (\ldots) \\
&\quad \longrightarrow H = [\mathsf{nodelete}(3)], e = \mathsf{delete}(\ldots);\ \mathsf{permute}(\ldots);\ \mathsf{recode}(\ldots) \\
&\quad \longrightarrow H = [\mathsf{match}(1,1)], e = \mathsf{delete}(\ldots);\ \mathsf{permute}(\ldots);\ \mathsf{recode}(\ldots) \\
&\quad \longrightarrow H = [\mathsf{match}(1,2)], e = \mathsf{delete}(\ldots);\ \mathsf{permute}(\ldots);\ \mathsf{recode}(\ldots) \\
&\quad \longrightarrow (\ldots)
\end{aligned}$$

The tree has only one root node which was generated without any constraints. The generated expression is the best guess by datadiff, which drops the `Count` column, rearranges columns and recodes (incorrectly) the `City` column. The nodes at the second level allow the user to add a number of constraints.

The first two sub-nodes prevent datadiff from generating two patches that it employed. The first one disallows recoding the column `City`. The resulting expression drops `Count` and rearranges the columns, which is the result we were hoping to obtain. The second sub-node disallows deleting the `Count` node. In this case, datadiff has to find other ways of reconciling the data sets, perhaps by recoding `Code` and dropping another column. Finally, the last two sub-nodes allow us to explicitly specify a mapping between columns. The listing only shows the first two (map column 1 to column 1; map column 1 to column 2), but the AI assistant can generate the full list.

4.2.3 ColNet assistant for knowledge graph linking

The lack of semantics and context in datasets may hinder the application of data analysis tools to, for example, identify errors like wrong values. The semantics provided by a

Knowledge Graph (KG) may play an important role to address the challenges that are being faced in data wrangling tasks.

In AIDA we aim at creating a set of AI assistant to link tabular data to a KG. Tabular data to KG matching is the process of assigning semantic tags from KGs (e.g., Wikidata, DBpedia or Googles KG) to the elements of the table. This task however is often difficult in practice due to metadata (e.g., table and column names) being missing, incomplete or ambiguous. Tabular data to KG matching tasks typically include:

1. cell to KG entity matching,
2. column to KG class matching, and
3. column pair to KG property matching.

We have designed and implemented ColNet [3], a column type prediction system relying on Convolutional Neural Networks (CNNs), semantic embeddings and KGs.

ColNet's goal is to identify the most accurate semantic type (*i.e.*, a KG graph class) or set of types for the columns in a table. ColNet use transfer learning techniques to minimize the impact of cells without a correspondence to the KG or an ambiguous correspondence (e.g., Apple, Virgin).

Samples in ColNet are constructed by concatenating the vector representation of the words in the labels of a set of entities of size h (i.e., the sample size). This way each sample is embedded into a matrix $\boldsymbol{x}(\boldsymbol{s})$:

$$colnet$$

$\boldsymbol{x}(\boldsymbol{s}) = v(word_1) \oplus v(word_2) \cdots \oplus v(word_n)$where $v(\cdot)$ represents d dimension word representation and $\oplus$ represents stacking two vectors. For example, considering a sample composed of DBpedia entities
`dbr:Bishopsgate_Institute` and `dbr:Royal_Academy_of_Arts`, its matrix is the stack of the word vectors of "Bishopsgate", "Institute", "Royal", "Academy", "of", "Arts" and two zero vectors, where we assume n is fixed to 8.

ColNet trains a CNN model $\mathcal{M}^{st}$ for each (candidate) KG class st and predicts a score p_k^{st} in $[0, 1]$ (for every class or semantic type st) given as input a set of test samples $\boldsymbol{x}(\boldsymbol{s}_k)$. ColNet eventually averages all the scores as the prediction score for the class st:

$$p^{st} = \frac{1}{N} \sum_{k=1}^{N} p_k^{st}$$

In our AI assistant definition:

- D: a CSV file
- e: the expression semType(c_i, st) assigns a specific semantic type st to an entity column c_i of a CSV file D
- Q: $Q(D, e) = p^{st}$, for test samples $\boldsymbol{x}(\boldsymbol{s}_k)$ in the target column c_i given $e =$ semType(c_i, st)

- $p_H(e)$: is the prior on the set of teh defualt (hyper)parameters of the AI assistant. For example, the target KGs, the target set of candidate classes (semantic types), the word embedding model, the sample size h, vector dimension d, number of words n, the number of samples N, the threshold α with respect to the score Q to filter out non relevant classes.

- H: As human intervention or constraints (*e.g.*, choices), the user could restrict the target KGs and the set of candidate classes, the user could also provide labeled samples for concrete cells like *Apple* is a *Company*, or provide labelled datasets from previous years. The AI assistant can also ask for concrete cases (*e.g.*, non clear cut cases, like *Apple* a *Fruit*). Ultimately, the user could choose the best semantic types from the suggested ones.

- $best_{\mathcal{D}}(H)$: is a set of expressions $e = \mathsf{semType}(c_i, st)$ for each entity column c_i in D

- f is a (enrichment) function that takes one or more expression e (*e.g.*, $best_{\mathcal{D}}(H)$) and a CSV file D and returns a JSON file with the semnatic type annotations for each column.

Although ColNet annotates columns in isolation, we use the whole CSV file as input dataset D as newer versions of ColNet already consider the context of surrounding columns. Note that, ColNet currently focuses on entity columns, that is, columns storing strings and possibly mentioning entities like person names, company names, etc. PType (Section 4.3.2) could first assign the column datatypes and provide ColNet with this information.

4.3 Bayesian AI assistants

Majority of AI assistants fits the optimization framework introduced in Section 3.3, but for some of them, this framework hides further useful information. In particular, the use of $\arg\max$ means that we only obtain one expression, while many assistants can provide a range of possible suitable expressions. In this section, we look at the Bayesian case where the solution is a probability distribution over expressions rather than just a single expression. We introduce the general approach in Section 4.3.1 and then discuss two concrete examples.

4.3.1 AI assistants through the Bayesian perspective

An alternative to the optimization view of an AI assistant, is to consider a Machine Learning approach to an AI assistant. Defining e as the set of expressions that perform some cleaning operation on a dataset D, the main idea is of a Bayesian AI assistant is to model the posterior distribution of those expressions given the data $p_H(e|D)$, where H accounts for any possible human intervention.

By defining this posterior distribution instead of a cost function $Q_H(D, e)$, these assistant are able to provide a richer knowledge over the possible transformations on the data.

For example, in the optimization based assistants, the optimization procedure was defines as:

$$best_{\mathcal{D}} = \arg\max_{e \in E_H} Q_H(\mathcal{D}, e)$$

In the Bayesian perspective, this is equivalent to defining $Q_H(\mathcal{D}, e) \equiv p_H(e|D)$ and computing the Maximum A Posteriori (MAP) of $p_H(e|D)$. However, by providing the full posterior of the expressions instead of a "best" outcome we are able to identify cases such as, e.g., multiple modes on the posterior with similar probabilities and we are able to provide uncertainty measures over the proposed transformations.

Additionally, we can write:

$$p_H(e|D) \propto p_H(D|e) p_H(e)$$

where $p_H(D|e)$ is the likelihood of the data under a specific expression e and $p_H(e)$ is the prior distribution of these expressions. Notice how the human interaction can be defined both in changes over the prior over the expressions and the likelihood model of the data, adding more flexibility to the types of human interactions over these assistants.

In the next sections, we cover two AI assistants that can be represented from a Bayesian perspective.

4.3.2 PType assistant for probabilistic type inference

ptype [2] is a probabilistic type inference method that can robustly annotate a column of data. As an AI assistant, it can be used to infer the data type (e.g. Boolean, integer, string, date, float, etc.) for each column in a table of data, as well as to detect missing data, and anomalies. The details of the notation and the model can be found in the ptype paper. Here, we describe the outputs of ptype, and how a user can interact with the corresponding AI assistant.

We first describe the output of ptype used for the column type inference, which is the posterior probability of the types denoted by $p(t|\mathbf{x})$. This represents our belief over the types given a data column **x**. We currently annotate the column type as the one with the maximum posterior probability value, which denotes the most likely column type. Note that one can apply a different decision strategy on $p(t|\mathbf{x})$ to determine the column type. In case of an uncertainty, i.e. when the distribution is not heavily on one data type, we could offer the user a choice of the most likely types or we could annotate the column with multiple possible types

Once the column type is inferred, we can use ptype to determine entries that do not conform with the inferred column type, e.g. missing data, and anomalies. The output now is the posterior probability of the i^{th} row type given the column type t and a data entry x_i, denoted by $p(z_i|t = k, \mathbf{x})$ assuming the column type is inferred as the type k. One way of using this output is to categorize the entries into two groups as type and non-type, where we collect the missing data and anomalies in a single group named the non-type. However, one could use these outputs differently, e.g. treating missing data and anomalies separately, rather than treating them together as non-types.

We now present a formal definition of the ptype AI assistant in the framework describe earlier. Note that the notation is an ongoing work, and could change in the future.

- e: the annotations of the column type, missing data and anomalies.
- D: a column of data
- H: human interactions with the ptype are various. For example, one can set the data types to be considered as column types in the column type inference. Moreover, a user can modify the model itself by specifying the missing data encodings that should be treated as missing data.
- Q : The objective function to maximize in the training of ptype is $\sum_{j=1}^{M} \log p(t^j|\mathbf{x}^j)$, where $\mathbf{x}^j$ and t^j respectively denote the j^{th} column of a given data matrix X and its type, and M is the number of columns. However, once the model is trained, we annotate test datasets using the outputs of the model $p(t|\mathbf{x})$ and $p(z_i|t = k, \mathbf{x})$, which can also be seen as scoring functions.
- f: the f function takes an expression e and a data column D and returns a data column with its annotations D^*.
- D^*: a column of data annotated in terms of data types. For example, this can include the column type, and type/non-type entries.

Lastly, the $best_D(H)$ and e depend on the decision rules used to determine column type, missing data, and anomalies. Assume we determine the column type as the one with the maximum posterior probability value, and label entries as type/non-type by grouping missing data entries and anomalies together. In this case, e consists of the inferred column type and a mask vector to denote non-type entries.

4.3.3 HIVAE assistant for filling missing data

The HIVAE [6] assistant helps the user fill missing data in a dataset based on observed data. It assumes that the missing data has the same distribution as the observed data. It then trains a deep neural network based on the observed data and uses it to generates appropriate values for rows with some missing data. We write X^o for the observed data, i.e. rows that contain data for all columns and X^m for the missing data, i.e. rows that are missing values for some of the columns.

The main idea of HIVAE is to maximize an Evidence Lower Bound (ELBO) on the observed data $\mathcal{L}(X^o; \phi, \theta)$, approximating its joint distribution. HIVAE is composed by a encoder layer $q_\phi(z|X^o)$, that models the latent space representation of the observed data, and a decoder layer $p_\theta(X^o|z)$, that models the likelihood distribution of the observed values in the data, given the latent space representation z.

The human intervention part of HIVAE is currently restricted to changing neural network hyper-parameters. These changes will affect the structure of the networks, and consequently, the optimization of the weights of the neural networks. Another possible intervention would be changing the likelihood model for a certain type of attribute (e.g. positive real attributes modelled as a Gamma distribution or a Log-Normal distribution). This is not part of the model at the moment, there is a one-to-one mapping between the type of the attribute and the likelihood model employed.

The training part of the model consists of optimizing the parameters of the DNNs (ϕ, θ) that maximizes $\mathcal{L}(X^o; \phi, \theta)$:

$$\phi^*, \theta^* \quad = \quad \arg\max \mathcal{L}(X^o; \phi, \theta)$$

The missing data imputation part is provided by a two step process, where we obtain a latent representation of the observed data using the encoder, and impute possible values for the missing entries using the decoder.

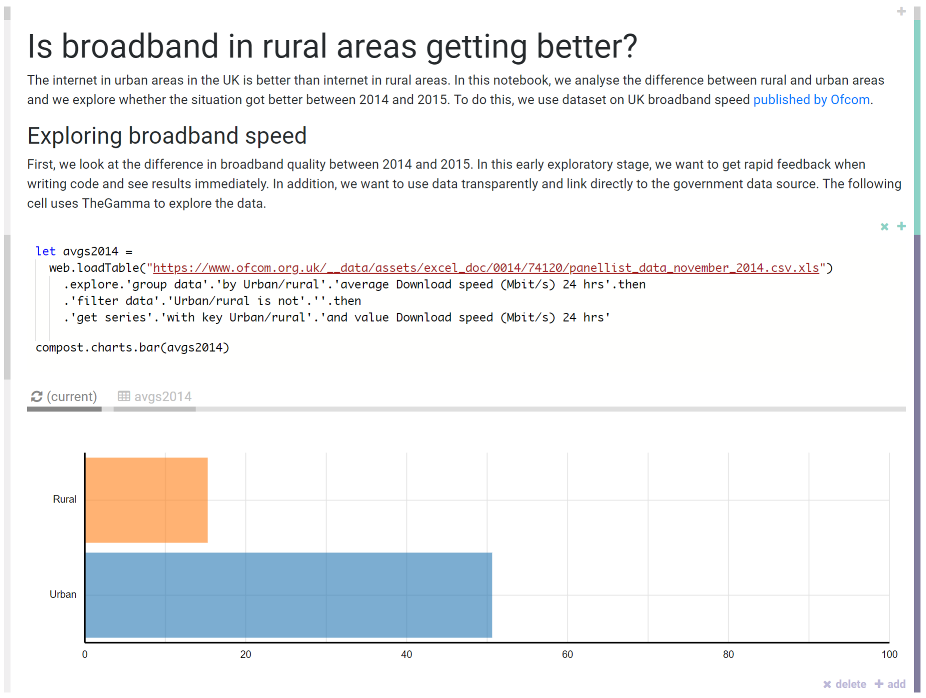
$$\begin{aligned} z &\sim q_{\phi^*}(z|X^o) \\ X^m &\sim p_{\theta^*}(X^m|z) \end{aligned}$$

We now define all these terms and relate them with the AI assistants

- D: In HIVAE, the input data includes a) the observed dataset X^o, b) the indicator mask for the missing values and c) the types of each attribute in the data.
- e: these would simply be the parameters ϕ and θ that constitute the encoder and decoder layers in the HIVAE model. Note that we also need to consider another set of parameters here, the neural network hyper-parameters ψ (dimension of the latent spaces, dimension of the MLPs, etc). These parameters can be changed by the user, but we are not optimizing them, at most, they are hand-tuned.
- Q: the cost function in our model is the ELBO $\mathcal{L}(X^o; \phi, \theta)$. This ELBO includes both the parameters of the layers that need to be optimized as well as those parameters that can be hand-tuned by the user, as described before.
- H: the human intervention part of HIVAE captures the neural network hyper-parameters ψ.
- f: the evaluation function is simply the imputation procedure of the HIVAE model. Once the model is trained, the imputation method for each attribute depends on the observed data and on the type of the attribute (e.g. imputing real attributes is done using the trained mean of the distribution, but imputing categorical values is done getting the argmax of the probabilities of each category.) Other alternatives, like multiple imputation are possible in this framework.
- D^*: this would be essentially the imputed data X^m, where we substitute the missing entries by the appropriate value provided by the imputation procedure selected.

5 AI assistants in data science tooling

In Section 3.4, we noted that having an abstract definition of AI assistant makes it possible to independently develop specific AI assistants for data wrangling tasks and tools that integrate such AI assistants in a data science workflow. We discussed several such AI assistants in Section 4. In this section, we discuss a prototype implementation of a support for AI assistants in a notebook system for data science.


Is broadband in rural areas getting better?

The internet in urban areas in the UK is better than internet in rural areas. In this notebook, we analyse the difference between rural and urban areas and we explore whether the situation got better between 2014 and 2015. To do this, we use dataset on UK broadband speed published by Ofcom.

Exploring broadband speed

First, we look at the difference in broadband quality between 2014 and 2015. In this early exploratory stage, we want to get rapid feedback when writing code and see results immediately. In addition, we want to use data transparently and link directly to the government data source. The following cell uses TheGamma to explore the data.

```
let avgs2014 =
  web.loadTable("https://www.ofcom.org.uk/__data/assets/excel_doc/0014/74120/panellist_data_november_2014.csv.xls")
    .explore.'group data'.'by Urban/rural'.'average Download speed (Mbit/s) 24 hrs'.then
    .'filter data'.'Urban/rural is not'.''.then
    .'get series'.'with key Urban/rural'.'and value Download speed (Mbit/s) 24 hrs'

compost.charts.bar(avgs2014)
```

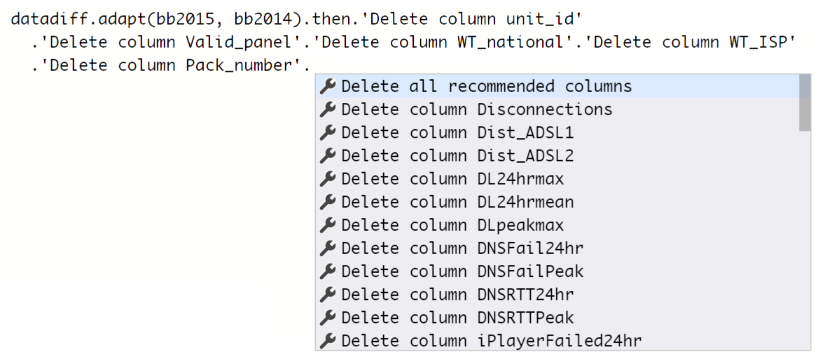
Figure 7: Wrattler notebook system with an example data analysis exploring the quality of broadband internet in the United Kingdom.

5.1 Wrattler notebook system

We implemented a prorotype support for AI assistants in the Wrattler notebook system [7]. Wrattler architecture differs from widely used notebook systems such as Jupyter and Rmarkdown [1, 5] in two key was which make it possible to support AI assistants:

- First, Wrattler extensions can be implemented as client-side components, running fully in the web browser. In contrast to Jupyter notebook with server-side kernels, this allows us to implement support for interaction in the notebook user interface.
- Second, Wrattler keeps all data in a separate data store. AI assistants can read data from the data store and write results to it, which means that they can be independent of any other programming language (such as R or Python) used by a data scientists.

Figure 7 shows a sample analysis in the Wrattler notebook. The example illustrates the first aspect of Wrattler. The code in the example, which performs a simple data aggregation and produces a chart, is fully executed in the web browser. Thanks to our general definition of an AI assistant, support for AI assistants can be added to other tools, but they have to provide the above two features: interactivity and the ability to access data that an analysis is using.



```
datadiff.adapt(bb2015, bb2014).then.'Delete column unit_id'
  .'Delete column Valid_panel'.'Delete column WT_national'.'Delete column WT_ISP'
  .'Delete column Pack_number'.
```

Figure 8: The datadiff AI assistant running inside Wrattler, a notebook-like environment for data science. The completion list shows data cleaning operations offered by an AI assistant and the preview shows the result of running the provided cleaning script (adapted from Figure 1).

5.2 Interactive extensions to Wrattler

In our prototype, we supported AI assistants using a data science scripting language The Gamma through a mechanism known as *type provider* [8, 10]. In The Gamma, scripts can be constructed by starting with input data such as broadband2014 and then typing "." which shows members and operations that can be accessed or invoked on the input data. The user then chooses one of the offered members and repeats the operation. A *type provider* is a small script that interacts with the user and offers available members and operations.

The example in Figure 7 shows a data aggregation script written in The Gamma, using one of the built-in type providers. The user invokes web.loadTable to obtain the initial data set (from a live URL). They then choose the explore member and choose from the offered choices – through the simple interaction of typing "." and choosing a member, the user is able to construct a data exploration script that involves operations such as grouping (by a column named Urban/rural) and calculating average speed (using a column named Download speed (MBit/s) 24 hrs).

5.3 Datadiff type provider

Figure 8 shows the simple datadiff AI assistant described in Section 4.1.1 integrated through the type provider mechanism in Wrattler. To invoke the assistant, the analyst first needs to specify the two input datasets. This is done using datadiff.adapt(d_1, d_2). This corresponds to invoking the $best_{\mathcal{D}}$ operation with input data $\mathcal{D} = (d_1, d_2)$ on an empty human interaction H_0. When the user types "." to see available members, the AI assistant is invoked and asked to generate a list of choices (shown in a drop-down box in the example). This is done by invoking the $choices_{\mathcal{D}}$ operation on the current trace of human interaction.

In our prototype, the code that the user sees directly corresponds to the human interactions performed when using the AI assistant. Operations such as Delete column WT_ISP that the user selects from a drop-down are used to construct a set of accepted patches, which is the H as defined in Section 4.1.1. One limitation of our current prototype is that it does not show the best expression (cleaning script) that the AI assistant generates, but merely evaluates it to show the clean data, as illustrated in Figure 1.

6 Future work and conclusions

6.1 Future work

In this report, we describe interactive versions of a number of data wrangling tools. Our main contribution is that we describe a unified way of implementing and using such tools. In this section, we identify several areas for future work – many of those require further examples of AI assistants before they can be answered with certainty.

Unifying data accress assistants. As noted in Section 3.1, our current definition of an AI assistant treats input data $\mathcal{D}$ as assistant-specific definition. Most of our assistants work with one or more tabular datasets containing various values and additional metadata annotations. Having a unified definition of data that is shared across all assistants would make integration of different assistants easier, but further examples are needed before we can propose a unified definition that is general enough.

From accountability to transparency. For many of our AI assistants such as Clever CSV (Section 4.2.1) or datadiff (Section 4.2.2), the expression e that is returned by the AI assistant is a human-readable data cleaning script. However, this is not yet the case for all assistants – for example, the current definition of the assistant for filling missing values (Section 4.3.3) defines e as the inferred parameters of a model – this makes the assistant reproducible, but not human-readable. An important challenge is finding alternative designs of such AI assistants such that the expressions e contain more understandable representation of the model.

Composing AI assistants. As noted in Section 1, data analysts will typically interact with many assistants over the course of an analysis. An intriguing question for future work is whether we could help the analyst not just during individual wrangling tasks, but also help them compose the available assistants most effectively. Combining assistants could also improve the quality of recommendations that they give – for example, a CSV parsing assistant (Section 4.2.1) might prefer a different way of parsing input file if it allows better semantic inferrence in an assistant such as ColNet (Section 4.2.3)

6.2 Conclusions

Data wrangling remains a tedious, time-consuming and manual task that is difficult to fully automate. We believe that a major advance in data wrangling is best achieved by developing semi-automatic methods that allow us to effectively combine automation and scalability of machine learning methods with human insight. In this report, we introduced the idea of an *AI assistant*, which captures the essence of the human-computer interaction pattern required for data wrangling.

An AI assistant assists the data anlysts in constructing a data wrangling script. It is *semi-automated* in that it runs in a loop that repeatedly performs an automatic optimization and asks for user input to influence the next iteration of optimization. It is *accountable*, meaning that it does not just turn dirty data into clean data, but it also produces scripts that are reproducible and often human-readable. Finally, it is *tooling-rich* in that it supports interactive programming mechanisms such as auto-complete in modern development tools like Jupyter.

In this report, we give a formal definition that captures the structure of an AI assistant and use it in two ways. First, we describe a framework of constructing AI assistants as a constraint-based optimization problem. Second, we describe a mechanism for integrating any AI assistants into a programming tool. Finally, we give a number of concrete AI assistants that address a wide range of data wrangling challenges, including data parsing and reconciling of multiple files, inferrence of types and semantic information and also filling of missing data.

References

[1] J Allaire, Joe Cheng, Yihui Xie, Jonathan McPherson, Winston Chang, Jeff Allen, Hadley Wickham, Aron Atkins, Rob Hyndman, and R Arslan. rmarkdown: Dynamic documents for r. *R package version*, 1:9010, 2016.

[2] T. Ceritli, C. K. I. Williams, and J. Geddes. ptype: Probabilistic Type Inference, 2019. Submitted for publication.

[3] J. Chen, E. Jimenez-Ruiz, I. Horrocks, and C. A. Sutton. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

[4] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 2003.

[5] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[6] Alfredo Nazabal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling incomplete heterogeneous data using vaes, 2018.

[7] T. Petricek, J. Geddes, and C. A. Sutton. Wrattler: Reproducible, live and polyglot notebooks. In *10th USENIX Workshop on Theory and Practice of Provenance (TaPP 2018)*, 2018.

[8] Tomas Petricek. Data exploration through dot-driven development. In *Proceedings of ECOOP*, volume 74. Schloss Dagstuhl, 2017.

[9] C. A. Sutton, T. Hobson, J. Geddes, and R. Caruana. Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling. In *24th ACM*

SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2018), 2018.

[10] Don Syme, Keith Battocchi, Kenji Takeda, Donna Malayeri, and Tomas Petricek. Themes in information-rich functional programming for internet-scale data sources. In *Proceedings of DDFP*, pages 1–4. ACM, 2013.

[11] Gerrit J. J. van den Burg, Alfredo Nazabal, and Charles Sutton. Wrangling messy csv files by detecting row and type patterns, 2018.