Can language models automate data wrangling?

Gonzalo Jaimovitch-López, Cèsar Ferri, José Hernández-Orallo,
Fernando Martínez-Plumed ✉, and María José Ramírez-Quintana

VRAIN. Universitat Politècnica de València, Spain
{gonjailo,cferri,jorallo,fmartinez,mramirez}@dsic.upv.es

Abstract. * The automation of data science and other data manipulation processes depend on the integration and formatting of 'messy' data. Data wrangling is an umbrella term for these tedious and time-consuming tasks. Tasks such as transforming dates, units or names expressed in different formats have been challenging for machine learning because (1) users expect to solve them with short cues or few examples, and (2) the problems depend heavily on domain knowledge. Interestingly, large language models today (1) can infer from very few examples or even a short clue in natural language, and (2) can integrate vast amounts of domain knowledge. It is then an important research question to analyse whether language models are a promising approach for data wrangling, especially as their capabilities continue growing. In this paper we apply different variants of the language model Generative Pre-trained Transformer (GPT) to five batteries covering a wide range of data wrangling problems. We compare the effect of prompts and few-shot regimes on their results and how they compare with specialised data wrangling systems and other tools. Our major finding is that they appear as a powerful tool for a wide range of data wrangling tasks. We provide some guidelines about how they can be integrated into data processing pipelines, provided the users can take advantage of their flexibility and the diversity of tasks to be addressed. However, reliability is still an important issue to overcome.

Keywords: Data Science Automation · Data Wrangling · Language Models · Machine Learning Pipelines.

1 Introduction

Data wrangling refers to repetitive and time-consuming data preparation tasks, including the transformation of data presented in different formats into a standardised form for easy access, understanding and analysis. The (semi-)automation of these manual and non-systematic tasks can impact the costs of data preparation significantly. If language models (on their own or integrated within other systems) are able to solve a significant proportion of these problems in the next years, the transformative effect on society and the marketplace would be huge, given how widespread these formatting chores happen (from spreadsheet manipulation to data science projects) [21].

One key difficulty of some data wrangling problems such as standardising a field into a single format stems in the context of interaction [54]. For automation to be

*To appear in the Machine Learning Journal, 2023, please do not distribute

really useful, the amount of information given by the users and their degree of involvement must be low enough so that there is a net gain in the process. For instance, in a standardisation of dates, the tool should be able to infer the transformation pattern from very few examples (or no examples at all), and complete the rest automatically. The second challenge for data wrangling is that data manipulation operations are very different. One project may require the integration of measurement units from different countries, while another project may involve identifying the order of a level of studies variable collected for thousands of customers. In many cases, the domain is not very specialised. For instance, in a date field, the day can be the first, second or third number, and these numbers can be delimited by different symbols. However, dates happens in the myriad of different transformations that we can find on the Internet or any other non-specialised source. This knowledge is general, but critical for data wrangling. An Artificial Intelligence (AI) system based only on basic string transformations may never find the right solution given just one example without domain constraints or background knowledge. For instance, the transformations needed for dates are very different from those used for addresses or emails, but these are domains generally well-known by humans.

There seems to be a great potential in language models [5] for data wrangling precisely because they compress huge amounts of human knowledge about many different domains, and have recently shown reasonably good performance in contextualising this knowledge for few-shot inference [41,46,8,22]. It is then very important to determine whether language models could be used in the future for data wrangling tasks, and whether they get better as the number of parameters increase, a question subject to recent debate [4,53]. The applicability of language models for the automation of other parts of data science (including the machine learning pipelines) may also be affected by the progress in data wrangling, especially as we move towards more domain-dependent and more open-ended tasks, as shown in the quadrants of Figure 1 in [15].

In this paper we test experimentally whether language models can be used to solve typical problems in data wrangling, using different kinds of prompts. Some (few-shot) prompts will have input-output examples and a single input ending the prompt, for which the language model will have to provide the output as a continuation of the prompt (e.g., `Input: ‘marshap@gmail.com’ \nOutput: ‘marshap’\n\nInput: ‘alant@hotmail.com’\nOutput:`). For the transformation datasets, we compare the inference power of GPT-3 with other specialised tools on a benchmark of simple data wrangling problems. Other (zero-shot) prompts simply describe the question or give instructions directly, without the need of extra examples (e.g., `Is (’bronze’, ’gold’, ’silver’) an ordinal?`). A combination of few-shot and instruction-based prompts is also possible, and also some fixed examples in the prompt, as we will explore.

For this reason many data wrangling tools not using language models combine the available information in the examples given by the user with some domain knowledge (‘any information the learner has about the unknown transformation before seeing the examples’ [49]), in an attempt to reduce the hypothesis space. Different approaches have been proposed relying on the coupling of ‘few examples’ and ‘background knowledge’. One of them is based on Inductive Programming [25], learning transformations from very few examples by incorporating prior knowledge about the domain in a declarative way. This domain knowledge is used to reduce the hypoth-

esis space making the generalisation process effective even from very few examples. As this approach suffers from intractability when background knowledge becomes large, the use of ad-hoc domain-specific languages (DSLs) (see [12,58]) restricts the search space, and has led to the first commercial products such as Microsoft Excel with FlashFill [24]. Even with domain-specific languages, many constraints on the transformations are added to make things work, or very specific collections of built-in facilities or functions. For instance, Amazon SageMaker Data Wrangler[†] contains over 300 built-in data transformations. Other Data Analytics tools such as Trifacta Wrangler [33] even allow the user to define their own transformations. Many systems combine some of these ideas or apply ad-hoc optimisations [26,6,18,39,24,50,49]. On the other hand, in [11,10], general-purpose inductive programming systems can still be employed with domain-specific background knowledge that is selected or ranked from contextual information or meta-features about the examples to be transformed. Still, this background knowledge has to be added to the system.

While we will make some comparisons, it is not the goal of this paper to see for each and every task whether current language models are better than the specialised tools above. The great advantage of language models is their versatility, and the power of dealing with a wide ranging of data wrangling problems, provided the user (e.g., a data scientist) comes up with the right way of prompting the language model. It is then more important to understand how the operation with language models can be inserted into the data processing and analysis pipeline, rather than just comparing what tool is best at each specific task. This would not even be realistic because (1) the best prompts are not always available for general users, especially because different prompts are needed for different tasks and (2) comparing a general system against dozens of specific systems may be unfair when considering the learning curves and other costs associated with dealing with these tools. Of course, the analysis assumes that new generations of language models will be generally accessible and more sustainable in the ratio between performance and compute. The progress and initiatives in the past year ([52,44,57]) suggest more generalised access to powerful language models may soon become commonplace.

To our knowledge, this is the first paper analysing the potential of language models for data wrangling systematically[‡], determining the influence of the type of data wrangling task, the relevance of the semantic content, the size of the model, the type of prompt and the number of examples.

The paper is organised as follows. Section 2 presents the problem of data wrangling in the context of data science and other tasks that involve data manipulation, the diversity of these tasks, a taxonomy and an analysis of the role of semantic information. Section 3 sets the experimental goals, the batteries and metrics we will use (and how they correspond with the taxonomy), the language models, prompts and few-shot regimes we will use. Section 4 discusses the results for each battery, including some examples and in some cases comparisons with some other systems or baselines. Finally, Section 5 summarises the contributions and the limitations. It also gives some guidelines for a general use of language models in data wrangling and other data processing pipelines and closes with future work.

[†] `https://aws.amazon.com/sagemaker/data-wrangler/`

[‡] A preliminary version of this paper, only including the manipulation battery, was presented in [32].

2 Data Wrangling: Taxonomy of Tasks and the Role of Knowledge

Many daily tasks that involve computers entail the conversion of data from one format to another, so that an application can duly digest the data. In a discipline such as data science, where data takes centre stage, this is even more so. It is widely recognised that a large proportion of the data analyst's time will be taken up with data preparation and transformation challenges appearing in messy datasets, what is generally referred to as *data wrangling*. Nazabal et. al, [36] provide a comprenhensive taxonomy of such problems into three main groups: those issues related to *organising the data*; those related to *improving the quality of the data*; and those related to *feature engineering*. Each of these large groups of tasks is subdivided into specific tasks according to the nature of the data wrangling problem they face (see Table 1). Under data organisation we find data parsing, data dictionary, data integration and data transformation tasks, all focused on obtaining the best data representation for the tasks to be solved. Data quality tasks include canonicalisation, missing data, anomalies and non-stationarity tasks related to cleaning corrupted entries in the data. Finally, feature engineering is a more diverse group that includes a more diverse range of operation with the features, from simple combinations and non-linear mappings to more sophisticated operations, such as embeddings.

However, what determines whether a particular data wrangling task is a candidate for automation by language models? To approach this question we have to know what language model are and what type of interface we have with them. Language models are conceptually simple systems: they estimate the probability $p(y|x)$ of a given sequence of characters or tokens y following another sequence x, in the spirit of efficient coding [47]. Today, these models are usually based on large deep learning architectures such as transformers (attention-based architectures [56]), but they still estimate this same probability. They are trained over massive natural language corpora and hence exploit the extrinsic patterns borrowed from humans. However, beyond making plausible continuations following the inputs (the so-called 'prompts'), or as part of this capability, recent systems such as BERT [16], GPT-2 [42], GPT-3 [8], and PanGu-α [60] can also be employed as 'few-shot learners', trying to exploit intrinsic patterns in the prompt. Few-shot inference happens when the models are able to extrapolate from previous examples in the 'prompt', without being retrained or fine-tuned. Extensive experimental research [29,28,59,31] is showing remarkable extrapolations from small prompts.

Going back to Table 1, given a set of tabular data, which of these tasks can be performed at the level of columns (features) and/or instances (values)? If that is the case, the limited window (number of input tokens) of a language model (of the order of hundreds of tokens) could be sufficient for these tasks, which could be excellent candidates for automation by language models. Indeed, the state of the art of language models suggest they can be a promising tool for data wrangling precisely because they (1) capture a wide range of domain background knowledge, and contextualise it to the problem quite effectively, without the need of extra knowledge (e.g., we do not have to tell them that '23/12/2021' is a date), and (2) they not only infer from very few examples (e.g., pairs of date transformations '`Input: 23/12/2021, Output: 12-23-2021`'), but we can also add hints to the prompt to make few-shot

learning more effective, or even zero-shot learning possible (e.g., '`The conversion of 23/12/2021 into US format is:`').

All this makes prompt-commanded language models very versatile, because they can do many things by just choosing an appropriate prompt. In this regime, they are able to perform a wide variety of 'few-shot' tasks when the prompt wraps several examples, which are 'continued' with textual data that can also contain transformed values or the answer to factual questions about the input data. Of course, many integration tasks (e.g., merging two tables) or those that require some sort of temporal analysis are not suitable (today) for this type of prompt-commanded AI systems, because of the size or structure of the data or the lack of memory of language models beyond what is expressed in the prompt.

Table 1. Taxonomy of data wrangling problems (adapted from [36]). Tasks categorised by the level of aggregation of the data to which they may be applied (table, feature or value). We indicate with • that most of the instances of the task may be automatable with language models, while ◦ represents that only some of the instances or variations may be appropriate.

Problem	Group	Description	Level			Automatable
			Table	Feature	Value	
Data Organisation	Data Parsing	Identify the structure of the raw data source so that it can be read properly (e.g., csv or xml files, relational databases, etc.)	•			
	Data Dictionary	Understand the contents of the data (e.g., profile of the data, meaning and type of each attribute, etc.)		•	•	•
	Data Integration	Combine related information from multiple sources (e.g., in different tables) into a single data structure (e.g., a table, or time series, etc.)	•			
	Data Transformation	Manipulate the shape of the data (e.g., switching the format of the table from a "wide" to a "long" format or vice versa) and extraction of relevant pieces of information from it (e.g., names of people or places, relationships, etc.)	•	•	•	◦
Data Quality	Canonicalisation	Standarise features and units obtaining a common representation (e.g., (e.g. U.K., UK and United Kingdom; or specific formats for dates, addresses, etc.)		•	•	•
	Missing Data	Detect missing entries and understand missing data patterns for repair (i.e., imputing those missing entries with other values according to different rules)		•	•	•
	Anomalies	Detect patterns that does not conform to expected normal behaviours (e.g., due to systematic errors in measurement devices or malicious activity or fraud)		•		•
	Non-stationarity	Detect changes in behaviour of data (e.g., dataset shift, protocol changes, etc.)		•		
Feature Engineering		Manipulate or create features based on existing ones (e.g., aggregating several features into a unique feature, one-hot encoding representations of categorical variables, etc.)		•		•

According to these considerations, Table 1 discusses good candidate tasks to be addressed by language models. However, we are also interested in the reasons why language models can make a difference in these tasks. The answer to this question is the high domain knowledge associated with them, their semantics. Knowledge is key in data transformation and cleaning, as well as other data-intensive tasks such as schema matching or data integration, data discovery, etc. [61]. For instance, automated data cleaning processes usually employ transformation and validation rules that depend on data types [33,43]. Most commercial systems [51,55,20] attempt to

detect semantic types, typically using a combination of rule-based approaches and dictionary lookup. However, these approaches are limited to a few data types or to those where it is possible to specify strict validations, and are often not robust enough to process dirty or missing data. For instance, a particularly difficult data wrangling task to automate is the semantic detection of ordinal data types, where the variables have natural, ordered categories, and thus a direction (e.g., `quality` $\in$ {`bad, average, good, excellent`}) and a myriad of variations of these labels –including typographical errors– depending on the source and situation.

Looking at those tasks in Table 1 indicated as automatable, we recognise the use of domain knowledge in all of them. Data dictionary tasks require some knowledge about the data many data wrangling tools simply lack (e.g., identifying that `undergraduate`, `postgraduate` and `PhD` are values of a data type that may represent study level). Data transformation also needs knowledge, to determine, for instance, that `12/18/2022` is a date that can only refer to 18th December 2022. Canonicalisation is even more clearly knowledge-dependent. For instance, statistical analysis does not suffice to tell whether `U.K.`, `UK` and `United Kingdom` are simply the same thing. Missing data imputation can be done through models, but on many occasions it depends on knowledge as well, such as imputing that the country for the city of Venice is Italy. Similarly, we know that a negative age is an anomaly that is clearly wrong, but an unusual negative temperature (in Celsius) might still be okay. Finally, for feature engineering, knowledge can do easily what learning representations may require enormous amount of data. For instance, we can only suggest that `density` is a more meaningful feature than `area` and `total population` because we know the semantics of the features. In the end, in all these examples there is a striking commonality: this is general knowledge that large language models have been able to capture and can use appropriately if prompted in the right way. This is what we want to explore in this paper.

The last column in Table 1 (column "Automatable") actually lists the data wrangling tasks we will analyse in this paper. The possibilities for automation will be illustrated with some experiments. Regarding the automatable tasks, those related to data cleansing, data quality, as well as the construction of new features, involve different types of transformation, standardisation, extraction or generation of information. These are easy to configure in input/output prompts (as strings), as we will see in the following sections. According to the taxonomy shown in Table 1, we will focus our study on five main tasks: First, *data transformation* involves the extraction of relevant pieces of information from multiple features, while discarding any unnecessary information. Also, we will analyse the automation of *canonicalisation* tasks where the objective is to standardise representations in characteristics, metrics and units. We will work on tasks for the detection and imputation of *missing values*; and, the detection of *anomalies* in the data that do not fit the normal patterns. Finally, we will see examples of the process of selection, manipulation and transformation of raw data into new features with different examples (*feature engineering*).

Overall, in this work we completely cover the set of problems in [36] that can be candidates for automation by language models, leaving out those tasks that, due to their size, access requirements and temporal nature, cannot be addressed by language models at present.

3 Experimental Design

Data wrangling can appear in many different moments of the data processing pipeline and can be handled by very different users, from non-expert users to advanced data scientists. The fundamental element of data wrangling is its non-systematic occurrence, and so are their solutions. When data wrangling tasks are identified in isolation, e.g., following Table 1, then some tools can use specific procedures for each of them. However, as a result, many variations or non-standardised data wrangling problems are left out of the range of these tools. In this paper, we want to study how language models can be used in a flexible way to attempt any data wrangling task, trying to emulate a scenario where a user has general access to an off-the-shelf language model and has some practice writing prompts. This mimics the situation of a programmer that has to solve many different problems by writing code with the same programming language. Determining what programming languages and environments can lead to effective solutions more easily is hard to evaluate, but this should not be an excuse to criticise the efforts to produce imperfect, yet still valuable assessment methodologies. This rationale guides our experimental setting.

Our experimental goals are: (1) determine to which extent a state-of-the-art language model can obtain good results on those data wrangling problems in Table 1 under the few-shot setting, (2) analyse the effect of the number of instances given in the few-shot setting, (3) explore the effect of the number of parameters of the language model to better understand the future potential, (4) analyse the performance of a state-of-the-art language model on those data wrangling problems in Table 1 under the zero-shot setting using different prompts; (5) study the variation of performance for different batteries and domains, and (6) compare the results with some other systems specifically designed for data wrangling.

3.1 Batteries and Metrics

For the experimental setting, we employ five batteries of data wrangling problems. Some of them were collected in previous studies. These allow us to compare the results given by language models with some other (data wrangling) tools. Other batteries have been collected for this paper. In this case, we have taken datasets from very different sources with the main criterion of diversity for inclusion. The five batteries that we use are:

Manipulation Battery

This battery contains several common data wrangling problems (see, e.g., [18]) of very different domains that require to convert an input into an output that has to meet a standard or canonical format, or extract part of it. This battery is built over the most comprehensive benchmark for data-wrangling transformation problems to date, the Data Wrangling Dataset Repository[§] [11], which we have extended considerably[¶] [7].

[§] http://dmip.webs.upv.es/datawrangling/

[¶] https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/mult_data_wrangling

The tasks are mostly of the Data Transformation, Canonicalisation and Feature Engineering groups in Table 1. Overall, the battery contains 117 different tasks divided into 7 different domains (*dates*, *emails*, *freetext*, *names*, *phones*, *times* and *units*). For every task we have 32 examples (we use 3744 instances in total) composed by an input string and an output string, and performance is evaluated with average accuracy: exact matching with the correct output. We provide further details about the tasks in each domain in Table 2 and some illustrative examples in Table 3.

Table 2. Manipulation Battery: Datasets included in the data wrangling repository and extended in this battery.

Task Description	Expected Output
Add Punctuation	The date in numeric format split by a punctuation sign
Change Format	The date in one particular format
Change Punctuation	The date in one particular format
Get Day	The day in numeric format
Get Day Ordinal	The day in numeric ordinal format
Get Month Name	The name of the month
Get Week Day	The name of the weekday
Reduce Month Name	The name of the month reduced to three letters
Set Format	The date split in DMY format
Generate Email	An email account created with the name and the domain
Get After At	Everything after the at symbol
Get Domain	The domain before the dot
Before At	Everything before the at symbol
After Symbol	Everything after a symbol
Between Symbols	Everything between a pair of symbols
Delete Punctuation	Remove punctuation
Delete Spaces	Remove blanks
Digit to End	Everything after the first digit if exists
First Character	Get first character
Get After Comma	Everything after a comma
Get Caps	Capitalise each word in a text
To Upper	Convert text to upper case
Add Title	The name with a title
Get Title	The title attached to the name, if exists
Generate Login	A login generated using the name
Reduce Name	The name reduced before the surname(s)
Add Prefix by Country	Phone numbers with the prefix of the countries
Delete Parentheses	The list of phone numbers without parentheses
Get Number	A phone number presented in the string, if exists
Set Prefix	The list of phone numbers with the prefix
Set Punctuation	A phone number split by a punctuation sign
Add Time	The time increasing the hour by the integer
Append o'clock Time	The time appending an o'clock time
Append Time	The time appending the integer as new component
Convert Time	The time formatted to 24 hours format
Convert Time	The time formatted to a given format
Convert Time	The time formatted to 12 hours format
Convert Time	The time changed from the first time zone to the second
Delete Time	The time deleting the last component
Get Hour	The hour component
Get Minutes	The minutes component
Get Time	A time presented in the string
Convert Units	The value transformed to a different magnitude
Get System	The system represented by the magnitude
Get Units	The units of the system
Get Value	The numeric value without any magnitude

Types Battery

The types battery deals with semantic type detection tasks, which are mostly of the Data Dictionary group in Table 1. Therefore, this battery aims to automatically detect the semantic data type of some columns in a given dataset. To build this battery, we

Table 3. Examples of data wrangling tasks of different domains included in the Manipulation Battery.

Domain	# Tasks	Example (*input* → *output*)
Dates	21	*74-03-31* → *31*
Email	10	*Jan.Kotas@litwareinc.com* → *litwareinc.com*
Freetext	25	*Association of Computational Linguistics* → *ACL*
Names	15	*Prof. Kathleen S. Fisher* → *Fisher, K.*
Phones	18	*John DOE 3 ... [TS]865-000-0000 ...* → *865-000-0000*
Times	24	*3:40 PM* → 15:40
Units	10	*12.20 dg* → *1220.0 mg*

followed a similar procedure to [30]. Firstly, we selected 11 semantic types from the DBpedia ontology[‖] (an ontology that describes semantic concepts extracted from web pages such as "Address", "Affiliation" and "Country"). Then, we collected 5 datasets from well-known machine learning repositories (Kaggle[**] and OpenML[††]) and selected from them those columns whose header names match the selected semantic types from DBpedia (such as *Age* or *Gender*) or are closely related to them (such as the header *ScheduledDate* which is related to the semantic type "date/time"). That allows us to use the semantic types as the real type labels for the columns. With all of this, the type battery is conformed by 17 columns. The list of datasets, the characteristics of the selected columns, and their (real) semantic types can be found in Table 4. Performance is evaluated by averaging accuracy by considering a success if the output given by GPT-3 contains the real semantic type of the column (in singular or plural).

Table 4. Types Battery: Datasets used for detecting the semantic type, with the number of numerical and categorical columns selected for the experiments and their real semantic types. In some datasets there are more than one column with the same semantic type. We used four datasets from Kaggle and one dataset from OpenML.

Dataset	# Examples	# Selected Columns		Semantic Types
		Numerical	Categorical	
MedicalNoShows	110527	1	4	**age, gender, time/date, neighbourhood**
GenderByName	148022	0	2	**name, gender**
CountriesWorld	227	0	2	**country, region**
SpeedDating	8378	1	3	**age, gender, race**
Zillow	10730	0	4	**city, state, metropolitan area, country**

Ordinal Battery

This battery is concerned with detecting and sorting ordinal attributes, tasks that are mostly of the Data Dictionary group in Table 1. This battery includes the identification of whether an attribute is ordinal or non-ordinal. In the case of being ordinal, we want to determine the order. For instance, `bronze`, `silver`, `gold`, and `platinum` should be identified as ordinal and given the order `bronze` < `silver` < `gold` < `platinum`. For composing the battery, we looked at the literature dealing with ordinal

[‖] `https://dbpedia.org/ontology/`

[**] `https://www.kaggle.com/`

[††] `https://www.openml.org/`

attributes, in particular [48,2]. These papers cover the attributes in the UCI datasets Cars, Nursery, BreastCancer, Hayes-Roth, Balance and CMC. In Hayes-Roth, Balance and CMC the attributes are represented by numbers, so we excluded these for being trivial. We also add some other datasets with a good number of categorical attributes with a higher proportion of non-ordinal cases, such as SoyBean and Mushroom, to have a more balanced battery of ordinal and non-ordinal attributes. All these were integrated into our battery. The full list of datasets and attributes, and their characteristics can be found in Table 5, and we will show all the attributes in Table 9. We will evaluate whether a system can distinguish between ordinal and non-ordinal attributes (just from their labels), summarised as accuracy, and then whether it orders them correctly (we will consider all the pairwise comparisons between attributes, aggregated into a single metric, Spearman correlation between the inferred order and the correct order).

Table 5. Ordinal Battery: Datasets used for ordinal attribute ordering, with the number of non-numerical non-binary features that are ordinal and non-ordinal in each of them.

Dataset	# Examples	# Features (non-numerical)	
		Ordinal	Non-ordinal
SoyBean	47	6	11
PostOperative	90	8	0
Nursery	12960	8	0
Mushroom	8124	2	16
Cars	1728	7	0
BreastCancer	699	4	1

Anomalies Battery

This battery, which deals with semantic outlier detection tasks, clearly corresponds to the Anomalies group in Table 1. It aims to detect the existence of values in the data that appear to be inconsistent with the remainder of that set of data. This is one of the hardest problems in data wrangling since anomalies are not normally encoded explicitly in the data. The related concept of outlier is purely statistical, but an anomaly may not be an outlier and vice versa. Outliers can be univariate (e.g., a person who is 2.2 meters tall), bivariate (e.g., in a survey of a human population, a 5-year-old is not an outlier and a person who weighs 90 kg is not an outlier, but a 5-year-old who weighs 90 kg is an outlier) or multivariate. Here we focus on univariate outliers or, more precisely, anomalies. Lots of methods exist to analyse numerical data that has outliers in it (see e.g., [3] for a good review), and much less for categorical data, most of them based on frequencies [27,13] where the rare values are usually (wrongly) treated as outliers. Anomalies can simply refer to a value that does not fit, not to their statistical frequency, such as having three rows with the value 'Umbrella' in a column of countries.

We will separate our analysis based on the type of attribute. For numerical attributes, as ground truth we will use boxplots to detect outliers (i.e., a data point that is located outside the whiskers of the box plot). For categorical attributes, given the difficulty of finding labelled anomalies, we will create synthetic anomalies by randomly altering attribute values, as it is done in related work (see, e.g., [14,34,9]).

Anomalies will be introduced in 1% of the values of each categorical attribute and randomly picked values of each of the other attributes of the dataset will be used for this purpose. For instance, if a particular dataset has 5 attributes and we are inserting anomalies in one of them, this process will be repeated 4 times inserting random values of the rest of the attributes, one in each turn. The idea is that the inserted anomalies have a different semantic meaning than the original attributes, and using values of the other attributes for this purpose is a straightforward solution that allows us to experiment with different types of values but in a similar context. Finally, for composing the battery, we looked at the literature dealing with detecting outliers in tabular data, in particular [1,40,37], which cover the attributes in the UCI datasets Wine, Ozone, Mpg, Iris, Glass, Ecoli and BreastCancer. The full list of datasets and attributes, and their characteristics can be found in Table 6. For evaluating performance, we will calculate the outlier detection hit rate per column of each dataset compared with the set of outliers proposed with those obtained using the boxplot method or the ground truth depending on the type of attribute.

Table 6. Anomalies Battery: Datasets used for semantic outlier detection, with the number of features with and without outliers. All features are numerical except for the dataset indicated by $^{\triangle}$, with categorical features.

Dataset	# Examples	# Features	
		w Outliers	w/o Outliers
Wine	178	7	5
Ozone	366	4	6
Mpg$^{\triangle}$	234	2	3
Iris	150	1	3
Glass	178	8	1
Ecoli	336	4	3
BreastCancer	699	6	4

Imputation Battery

The tasks here are mostly of the Missing Data group in Table 1, namely finding missing values in data and trying to impute these values. Traditionally, there are several ways of dealing with missing values [45,23,19]. Replacing the missing values by a fictitious value (imputation) is usually a better practice than ignoring or removing the row. The new value is computed by means of simple strategies such as employing the mean or median (for numerical values) or mode (for categorical values) of the feature. More sophisticated methods of imputation are used by estimating the value from the other attributes with predictive models. In order to test the utility of language models to impute missing values, we are going to employ different datasets. For the experiments, we consider three well-known datasets that are frequently used by the machine learning literature from the UCI repository [17]: Adult, Iris and Mpg. Additionally, we also employ databases with information in specific fields such as the UK Postcode Address FILE [‡‡], tennis players from ATP [§§], and a simplified version

[‡‡]Extracted from: `https://www.poweredbypaf.com/product/paf/`

[§§]Source: `https://datahub.io/sports-data/atp-world-tour-tennis-data`

of the UCS Satellite Database [¶¶]. These are three databases that can be seen as examples of specific domains. Apart from numerical and categorical features, in these three datasets we can find features that are textual. We believe these datasets are representative of real databases that could present missing data, and thus they can be especially useful to show the capacity of GPT-3 to impute data with respect to other classical predictive techniques, since classical imputation methods do not work correctly with textual features as an output (structured prediction models would be needed, or generators, which is why language models may be a good option). The list of datasets and attributes can be found in Table 7.

Table 7. Imputation Battery: Datasets used for imputing missing values, with the number of numerical, categorical, and textual features. We used three datasets from UCI and three databases with information from specific fields.

Dataset	# Examples	# Features		
		Numerical	Categorical	Textual
Adult	48842	6	8	
Mpg	234	6	0	
Iris	150	4	0	
PAF-Address	26	0	1	7
ATP Players	550	8	2	6
UCS-Satellite	4852	8	4	4

We will use a traditional imputation method for comparison. A *DecisionTreeClassifier* from the scikit learn library [38] with the default configuration using 99 rows of the table to train a model that is used to predict the missing value in another row (not included in the 99 rows).

For each example, we repeat the procedure 10 times, and we measure the performance of the imputation comparing the predicted value with the actual value. In the case of the categorical attributes, we show the mean accuracy in imputing missing values. For the numerical attributes, we divide the estimated mean absolute error (MAE) by the standard deviation (σ) of the values of the feature. To make it more comparable with accuracy, we calculate its complementary, i.e., $1 - \frac{\text{MAE}}{\sigma}$.

3.2 Language Models and Prompts

As we discussed at the beginning of this section, despite the taxonomy in Table 1, there are thousands of variants of data wrangling tasks, and success or failure may depend on formats, domains, extra data availability, and many other factors that make each situation unique. For instance, standardising addresses in an international context is very different from discovering the order of a feature expressing martial arts levels. Having hundreds of specific tools or domain-specific languages is not a scalable solution for this diversity. Accordingly, we want to consider data wrangling pipelines where a user has access to an off-the-shelf language model and plays with a few prompts to explore whether the specific data wrangling at hand can be solved. It is not our goal to find the optimal prompt for each task and language model, but some prompts that users (not necessarily expert data scientists) can come up for

[¶¶]Extracted from: `https://www.ucsusa.org/resources/satellite-database`

making this data wrangling process. We do not want to overfit to the best prompt for each task and language models, as both tasks and language models evolve and change constantly. We want to have a general understanding of areas of higher potential in terms of results versus the effort of thinking of a good prompt and the associated examples.

We use four versions of OpenAI GPT-3 [8] of increasing capabilities: Ada, Babbage, Curie and DaVinci which line up closely with 350M, 1.3B, 6.7B, and 175B parameters, respectively. We mostly focus on one architecture, GPT-3, since it is still considered state of the art and highly representative. Although there are other large language models in the literature, the access to them has issues about open access to the source code, the cost per token, the necessary infrastructure, the privacy of the APIs or public use of results. Some collaborative initiatives are starting to test other large language models to evaluate their capabilities, making evaluation data public, but access to the language models directly is limited. In particular, the BIG-bench collaboration [7] has trained and evaluated Google's latest language models (Big-G) [7], and we could include our Manipulation Battery. Although we do not have access to the models, we have access to the results up to 3-shot (see Figure 1). The comparison shows that GPT-3 is representative of the state of the art in language models. In particular, the most advanced GPT-3 model, Da Vinci, has very similar results to other top language models for this battery***.

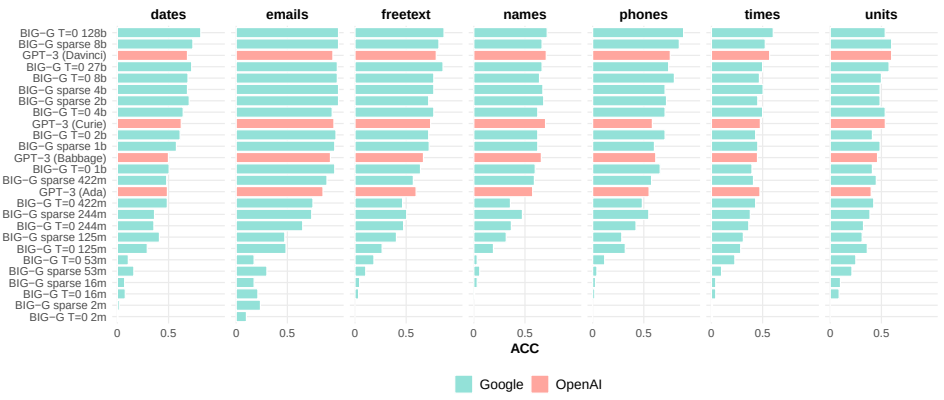

Fig. 1. Average accuracies per language model and domain on the Manipulation Battery tasks up to 3-shot. Language models sorted by average accuracy across all domains. The y-axis shows the id of the architecture (OpenAI GPT-3 or Google BIG-G models) and, in each case, whether the model is dense or sparse [62], and the number of parameters.

Focusing, therefore, on the use of GPT-3, we first analysed several possible prompts. Since our aim is not to find the optimal input prompt for each task and model, but rather to provide illustrative (simple) prompts, we explore a few choices that could be considered by a user that is familiar with language models and the prompt samples

***For the sake of replicability and reproducibility, all the code and results can be found in `https://github.com/gonzalojaimovitch/lm-dw`

that are recommended in the language models APIs[†††]. Also, in Table 10 we show the templates used and the alternatives we tried for some batteries which were discarded due to its low specificity (making it more difficult for GPT3 to understand the task) and, therefore, their poorer results in our initial experiments. In the end, what we want to show is that, in a simple way, any practitioner can use a pretrained language model to semi-automate many data wrangling tasks that appear in the data-processing pipeline, without the need to train predictive models or fine-tune pretrained models for each different task.

As a result, for each of the data wrangling tasks we will use different input prompts, trying to keep them as natural and simple as possible. Depending on the task, we may use different few-shot or zero-shot schemas. For instance, for the manipulation battery we will use a few-shot approach, while for the rest of tasks, we will not provide exemplars, but rather simple instructions about the task that we expect the language model to perform, thus following a zero-shot scenario.

Manipulation Battery
As all the examples in this battery are input-output pairs, prompts are easy to figure out here. Simply, the main prompt we will use follows an input-output style, where the string '`Input:`' is used to indicate the start of the input, and the string '`Output:`' is used to indicate the start of the output. The line break `\n` separates the input from the output of an example, as well as the examples in the prompt (when one or more examples are provided). The instance will have one (one-shot) or more (few-shot) input-output pairs. They will be randomly selected (without considering the possible order sensitivity of GPT-3 [35]) from the same problem and domain, and one single input will end the prompt. The language model will have to provide the output by continuing the prompt. Our intention is that GPT-3 generalises the concept only from the instances provided in new instances of the same task with no other information or description of the task at hand in the prompt. The prompts given below are two one-shot examples (from different domains):

```
Input: ‘290386’\nOutput: ‘29-03-86’\n\nInput: ‘250374’\nOutput:
```

```
Input: ‘08:50-09:30’\nOutput: ‘09:30’\n\nInput: ‘09:50-08:30’\nOutput:
```

Types Battery
In this battery we use two prompts to determine the type of a column. We follow a zero-shot strategy in that the first prompt asks the system for the "domain" that best describes a set of values randomly chosen from the column. Since "domain" is a broad term with several meanings, the second prompt directly asks the system for the "semantic type" of the selected values. Examples of each prompt are:

```
What is the best domain that describes the values in {2016-04-29T18:
38:08Z,2016-04-29T16:08:27Z,2016-04-29T16:19:04Z,2016-04-29T17:29:31Z,
2016-04-29T16:07:23Z,2016-04-27T08:36:51Z,2016-04-27T15:05:12Z,2016-
04-27T15:39:58Z,2016-04-29T08:02:16Z,2016-04-27T12:48:25Z}?
```

[†††]For instance, OpenAI collects many different examples: `https://openai.com/blog/openai-api/`

```
What is the best semantic type that describes the values in {male,
female,female,female,male,male,male,male,female,female}?
```

For the experiments, the number of values to be included in each prompt has been set to 10, as shown in the above examples. To mitigate the effect that the random selection of these 10 examples could have on the performance of the task, we have repeated the experiments with each prompt and column 5 times, and then, the results were averaged.

Ordinal Battery
Here we will try two different ways (prompts) to distinguish ordinal and non-ordinal features. In the first prompt we will ask the system if a given value is greater than another, repeating this for all possible combinations of values in each attribute of a dataset and computing whether the final order between all the unique values is consistent. An example of a prompt follows:

```
Is "house" higher than "apartment"? Yes\n Is "apartment" higher than "house"?
No\n Is "red" higher than "blue"? No\n Is "blue" higher than "red"? No\n
Is "old" higher than "young"? Yes\n Is "young" higher than "old"? No\n Is
"totally agree" higher than "agree"? Yes\n Is "agree" higher than "totally
agree"? No\n Is "New York" higher than "Chicago"? No\n Is "Chicago" higher
than "New York"? No\n Is "Heavy rain" higher than "Showers"? Yes\n Is "Showers"
higher than "Heavy rain"? No\n Is "gold" higher than "platinum"?
```

While this is a 12-shot, followed by the real question at the end (the 13th line), it does not really need any real example. The context is always the same for all the examples, while only the 13th line changes. This long context is added because it helps to frame the question and gives better results. We do compare all pairs in the attribute (and in both directions), and calculate the rank of each value depending on how many times it compared favourable against the rest. This gives us a 'rank' for each value. Taking this rank as the derived order, we check how many times the comparisons follow this order, and if this is greater than 75% then we say the attribute is ordinal, otherwise it is non-ordinal.

Alternatively, we will try an even simpler version of the prompt where we will directly ask if the unique values of a given attribute of a dataset are of ordinal type:

```
Is (low, medium, high) an ordinal? Yes\n Is (door, window, wheel) an ordinal?
No\n Is ('2', '4', 'more') an ordinal?
```

Again, the first two lines are fixed and only the last one changes depending on the attribute. This prompt is much easier, but does not give us an order, just whether the attribute is ordinal or not.

Anomalies Battery
Here we will follow a zero-shot strategy where we will provide the language model with a prompt asking directly whether *there are any outliers in* a given set of data.

For this battery we only include one prompt. We performed many preliminary tests to get good results. While we were looking for anomalies and not outliers, in the end we saw that the results were similar when we modified the prompt by asking for anomalies, oddities or abnormal phenomena in the data instead of using the word 'outliers'. A couple of examples of the prompt follow:

`Are there any outliers in {70°F, 71°F, ..., 74°F}?`

`Are there any outliers in {audi, chevrolet, dodge, ford, ..., volkswagen}?`

Imputation Battery
We use two prompts to make the language model infer the missing value from a set of examples. We use instances without missing values in the prompt and we leave the last line for the instance with the missing value. We have explored two alternatives. In the first one, *Full prompt*, we use all the available features in the data set. This is an example of the *Full prompt*:

```
City: Detroit, State: Michigan, County: Wayne \n City: Fargo, State: North
Dakota, County: Cass \n ..., City: Athens, State: Georgia, County:
```

The second approach, *1-Feature prompt*, is much shorter usually. We select the most useful feature[‡‡‡] For example, for the same dataset as above, if we determine `City` as the most relevant feature for inferring `County`, an example of the *1-Feature prompt* would be:

```
City: Detroit, County: Wayne \n City: Fargo, County: Cass \n ..., City:
Athens, County:
```

As we can see, the *Full prompt* strategy will make prompts very large (requiring many tokens from the language model) as soon as the instances have many features. Because of that we will employ a nine-shot approach, so only nine complete rows will be used. We will also explore a zero-shot approach with this configuration, which is simply the full row where the missing value appears. The second strategy (*1-Feature prompt*) allows us to provide more training examples without using too many tokens. Additionally, given that only one input feature is employed we need to increase the information provided to the language model, specifically we will employ 99 examples.

4 Results and Discussion

We start by analysing the performance of the different GPT-3 family of models (Ada, Babbage, Curie and DaVinci). The models are employed on the various data wrangling batteries described in the previous section. The result metrics are assessed using the different few-shot learning settings, including zero-shot regimes for some of the batteries, as explained in the previous section.

[‡‡‡]We compute the most useful attribute according to the attribute importance of a random forest model learned from the whole table using the attribute with the missing value as an output.

Let us start with the Manipulation Battery. In this case, the models are given several input examples in the 'input-output' prompt and no other further information or description of the task at hand. Figure 2 shows the results obtained by using the four different models (Ada, Babbage, Curie and DaVinci) and different few-shot learning settings, from zero-shot to ten-shot for all domains (which make a total of 11 configurations per task). Regarding the results, we see the sharp increase from zero-shot learning to 1-shot learning, and a more moderate increase that stabilises around 9-shot inference. In general, the results show that GPT-3 can be employed to learn simple transformations from few examples, and, as expected, the accuracy improves when we provide more instances. We also see that, as expected, the most powerful engine is DaVinci. Nevertheless, the performance is not uniform across the analysed domains. The domain *emails* is the one where the GPT-3 models obtain the highest performance, whereas *units* is the domain with the lowest performance. This may be related to the need of semantic information about the domain but also some reasoning or calculation capabilities (e.g., multiplication in the case of units). We include further details in the supplementary material: disaggregated results in Figure 8 and a set of illustrative examples of wrong answers obtained by GPT-3 for problems with different types of inputs (Tables 11 and 12).

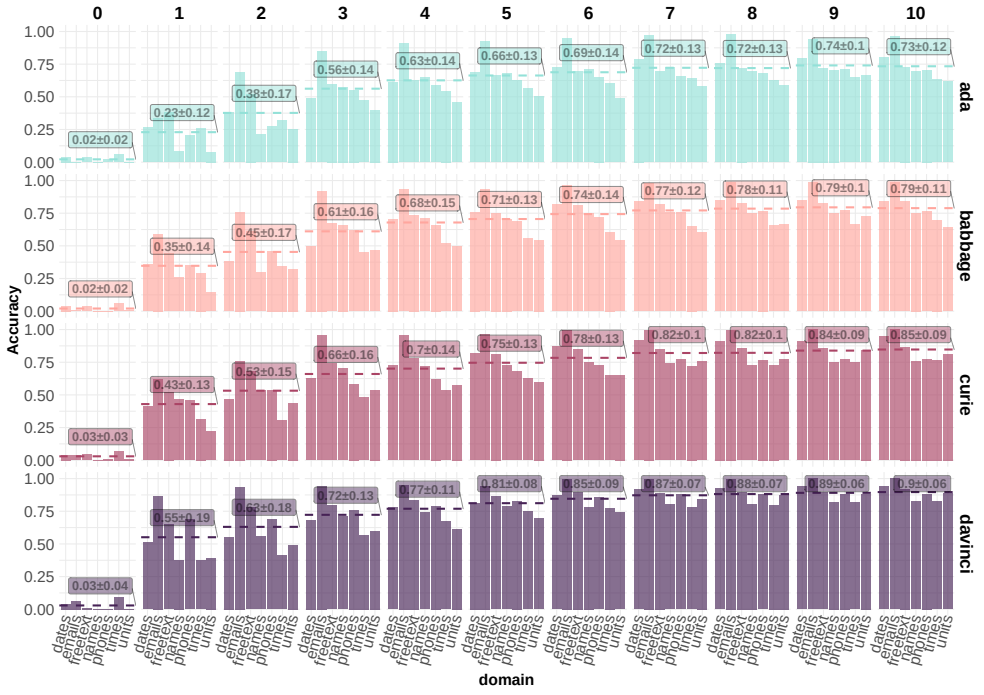

Fig. 2. Average results for the seven domains in the Manipulation Battery and the four versions of GPT-3. Each plot represents how many examples are given (from zero-shot to 10-shot). The dashed horizontal lines show the average results per system. Disaggregated results for all tasks shown in Figure 8 in the supplementary material.

With the intention of getting more insight into how the models fail, we perform a fine-grain analysis of the 'units' domain in the Manipulation Battery. Table 8 includes examples of some of these tasks to better understand the differences in performance shown in Figure 3. The problems in tasks *getUnits-i* and *getValue-i* (see Table 2 for details) can be translated as 'extracting a part of the string', a transformation that the GPT-3 models can solve. Hence, we see that GPT-3 presents good results in domains where tasks can be solved by simple string transformations. However, *getSystem-i* and *convert-i* are much more complex tasks. Thus, *getUnits-i* requires the identification of the unit acronym (e.g., 'cl' for centilitres) and relating it with its *dimension* (e.g., volume), while *convert-i* needs to perform an arithmetic operation (e.g., a division), in addition to the identification of the conversion coefficient to the target unit (e.g., a coefficient of 1000 to convert milligrams into grams).

Table 8. Examples of problems in the domain 'units'.

Problem	**Input → Output**
'getUnits-1'	56.77cl → cl
'getValue-1'	56.77cl → 56.77
'getSystem-1'	56.77cl → Volume
'convert-1'	1441.8mg; g → 1.4418001

Finally, in order to compare the performance of GPT-3 with other data wrangling systems, we consider the subset of 26 problems in the Manipulation battery, for which there are results in the literature. We make the comparison for the 1-shot setting, which is the same setting used by the other systems. We compare GPT-3 DaVinci and the following data wrangling tools: FlashFill [25], TrifactaWrangler [39] and Dynamic Background Knowledge (DBK) [11]. The results (displayed in Figure 4) show that general-purpose language models are competitive with first-generation data wrangling tools such as FlashFill, and are getting closer in performance to more sophisticated tools such as DBK. Again, we see that the performance of the compared systems is related to the types involved in the target functions. The best results are obtained in domains where the problems are solved by simple string operations, while in other domains like *units* where some functions include arithmetic operations the results are much worse. The exception is DBK, which can induce the domain of the problem and then select proper base functions to address it.

We now move on to the rest of the experimental batteries. Figure 5 shows the results obtained by using the most powerful GPT-3 engine, DaVinci, for the Types and Ordinal Batteries. It is interesting to note that there is, in many cases, a notable difference depending on the prompt.

For the case of the Types Battery, GPT-3 is more accurate the more informative the prompt is: asking for the "domain" seems to be less specific than asking for the "semantic type" of the values. In general, the results for the type detection using the second prompt are very satisfactory, with a mean success rate of 0.7 in front of a mean accuracy of 0.2 obtained with the first prompt.

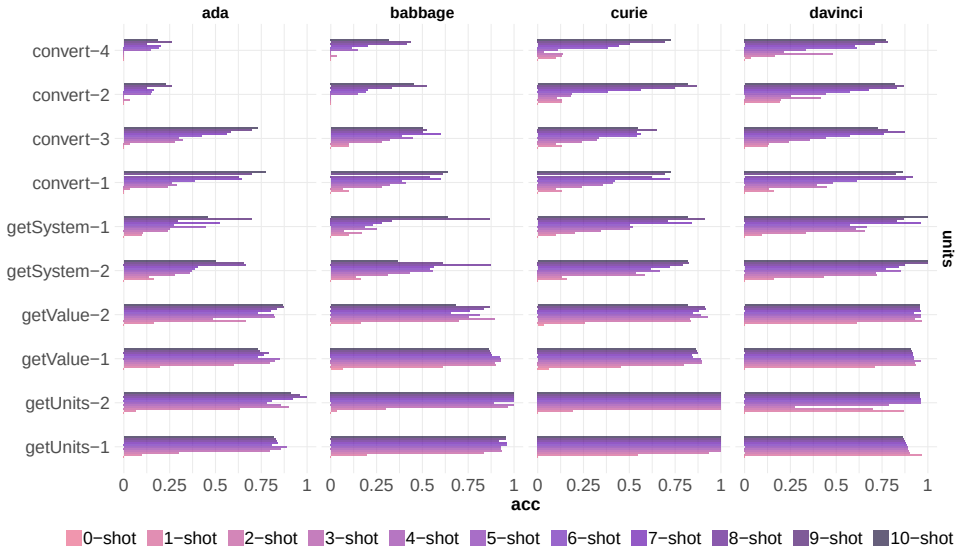

Fig. 3. Average accuracies for the tasks in the *units* domain for all GPT-3 systems and learning settings. Complete details of all domains and descriptions of all tasks are presented in Figure 8 and Table 2, respectively, in the supplementary material.

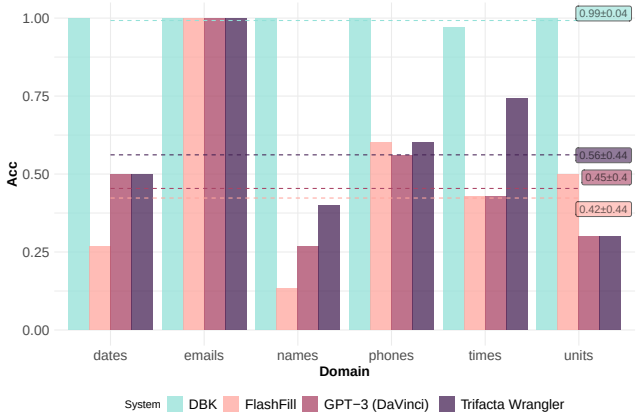

Fig. 4. Average accuracies of GPT-3 (DaVinci version), FlashFill [25], Trifacta Wrangler and DBK [11] for a 1-shot learning setting. Results of the compared systems are obtained from [11,10]. The tasks addressed are a subset of those in Figure 8. Coloured, horizontal lines show the average results per system across domains.

Let us analyse in more detail the answers given by GPT-3 for this battery. First of all, we must be aware of the expressive power of natural language, which implies that there is no unique way to name a concept. In terms of solving the semantic type task, this means that some values can be assumed as belonging to different but related types. For instance, *Palmira* and *Verona* could be considered as values of the type

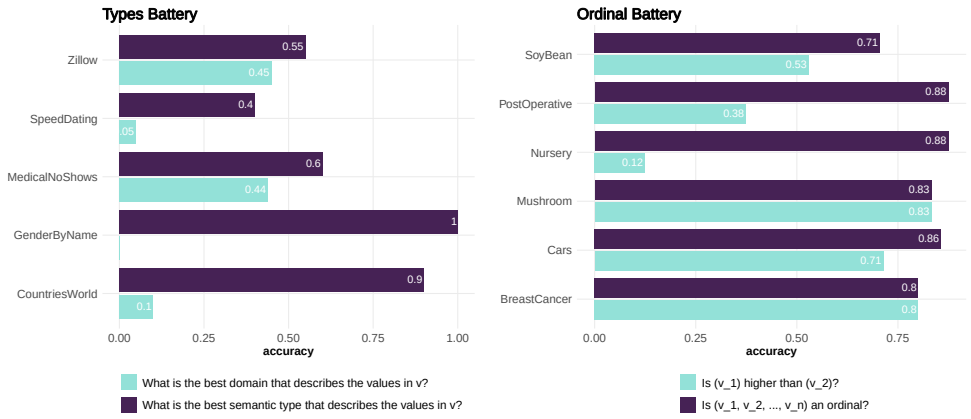

Fig. 5. Average results for different datasets and the types and ordinal batteries, using different prompts.

City, but also of the type Town or even Place. Although it can be argued that a city, a place and a town are not exactly the same concept (there exist some differences among them), it is clear they are related (all of them are locations). That is what we observe in the experiments, with GPT-3 giving all these answers for one of the columns of the Zillow dataset depending on the prompt, as shown in Table 13 in the appendix. Given that for evaluating the performance of the system we set the real semantic type of this column as "city", the rest of the answers were considered as failures. This fact explains the increase in accuracy we got with the second prompt since it directly asks for the "semantic type" allowing the system to focus on the least general concept (type) to which the observed values belong. Note that, "place" could be considered a much too general type for denoting *Palmira* and *Verona*, whereas "town" could perhaps be too much specific). Another way to solve the problem of having a set of possible answers (e.g., all possible column names with some a priori probabilities) would be to consider the conditional probabilities ("logprobs") provided by the language models (i.e., how likely some word can appear in the text given the other one in this text.) for each possible output and combine them with the a priori probabilities to determine the most likely column name.

Apart from these considerations related to the performance of the system on the Types Battery, we would like to highlight that the flexibility of GPT-3 providing several different answers as potential types is a feature rather than being a drawback. It is evident that language models can work as effective tools for solving this kind of Data Dictionary tasks, since they do not need (predefined) type ontologies to solve them. In fact, from a general point of view, a user could consider that any of the three answers given by GPT-3 for the above example are acceptable (i.e., they are valid types for the column), since with any of them the user is able to know that the values of such column are related to locations and not with other concepts such as names and countries.

Finally, the differences observed in the results of the Types Battery when GPT-3 deals with nominal and numerical attributes should be discussed. It is relatively easier for GPT-3 (and also for other language models) to infer the right type for a

nominal attribute than for a numerical one. The reason is clear; the values of nominal attributes are usually different (and specific) depending on the real concept (i.e., names, cities, countries, ...) to which the values belong to. However, it is much more difficult to determine whether a few numerical values such as {2, 4, 15, 23} correspond, for instance, to ages or Celsius degrees without any additional information. In the experiments we carried out, for both prompts, GPT-3 fails in determining that the semantic type of the two numerical attributes in the battery is "age", being "numbers" the most common type returned by the system (see Table 13 in the appendix). It could be interesting to explore prompts that include a description of the dataset domain, and observe whether with this information the language model is successful. For instance, if we give information that the table is about customers, then given some numbers, the language model could infer that the type might be "age".

In the case of the Ordinal Battery, it also depends on the prompt and domain. In general, however, asking whether there is an ordering between the unique values of an attribute (prompt 2) seems to be more effective. The results for telling ordinal vs non-ordinal are very satisfactory, with an average success rate of 0.83 for this prompt. However, if we look at how good the orderings are, the picture is a bit more elaborate. Table 9 shows all cases, with the Spearman correlation of the predicted ordering and the actual ordering using prompt 1, whether the method predicted ORDINAL or NON-ORDINAL with prompt 1 and prompt 2 and the actual value of ORDINAL/NON-ORDINAL. When the Spearman correlation is '-' the first prompt gave some unresolved comparisons, and the order could not be calculated. All these were assigned to NON-ORDINAL systematically.

In all those cases where an ordinal attribute is correctly classified as ordinal (14 out of 63) the average Spearman correlation is very high: 0.95. These are highly reliable cases where the order of attributes is perfectly determined (8 out of 14) or reasonably good (worst case is 0.78 correlation). Some of these attributes have a textual representation of numbers or intervals (e.g., age, tumor-size, etc.) or are very easy ('high', 'mid', 'low') so it is not surprising that language models do well. Many discrepancies not due to unresolved comparisons happen in cases where the correlations are high, but not high enough (e.g., inv-nodes, doors, children, has-nurs, health, housing, social, L-02), all of which are well categorised by the second prompt. In these cases we can say that recognising whether an attribute is ordinal or not is doable, but the order might not be good enough. Then there are some cases where the correlation is low and both prompts fail to recognise it is an ordinal (persons, population, form) or just the first prompt (parents, L-O2). Finally, there are some non-ordinal features that get high scores and are categorised as ordinal by one or both of the prompts (gill-spacing, habitat, leafspots-halo, mycelium, roots, stem-cankers). Some of these have an 'absent' value, which is usually recognised as having a lower order than other values, or other elements that could suggest that are partially ordinal.

Moving to the Anomalies Battery, and starting with the analysis of numerical attributes, Figure 6 (left) shows the poor performance of GPT-3, with a median success rate of 0.16. Some examples of its operation are shown in Table 14 in the appendix. Why this poor performance? This may be due to the fact that in many of these cases the role of semantic information is limited, and most especially as we

Table 9. Quality of orderings for the Ordinal Battery: Spearman correlation of the predicted ordering using prompt 1, the predicted types (with prompts 1 and 2) and the actual type.

Dataset	Attribute	Spearman	Pred. w. Prompt1	Pred. w. Prompt2	Actual
breastcancer	age	0.95	ORDINAL	ORDINAL	ORDINAL
	breast-quad	-	NON-ORDINAL	ORDINAL	NON-ORDINAL
	inv-nodes	0.80	NON-ORDINAL	ORDINAL	ORDINAL
	menopause	-	NON-ORDINAL	ORDINAL	ORDINAL
	tumor-size	0.90	ORDINAL	ORDINAL	ORDINAL
cars	buying	1.00	ORDINAL	ORDINAL	ORDINAL
	class	1.00	ORDINAL	ORDINAL	ORDINAL
	doors	0.70	NON-ORDINAL	ORDINAL	ORDINAL
	lug-boot	0.87	ORDINAL	ORDINAL	ORDINAL
	maint	1.00	ORDINAL	ORDINAL	ORDINAL
	persons	0.50	NON-ORDINAL	NON-ORDINAL	ORDINAL
	safety	1.00	ORDINAL	ORDINAL	ORDINAL
mushroom	cap-color	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	cap-shape	0.09	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	cap-surface	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	gill-attachment	-	NON-ORDINAL	ORDINAL	NON-ORDINAL
	gill-color	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	gill-spacing	1.00	ORDINAL	ORDINAL	NON-ORDINAL
	habitat	0.93	ORDINAL	NON-ORDINAL	NON-ORDINAL
	odor	0.28	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	population	0.14	NON-ORDINAL	NON-ORDINAL	ORDINAL
	ring-number	0.87	ORDINAL	ORDINAL	ORDINAL
	ring-type	0.38	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	spore-print-color	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	stalk-color-above-ring	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	stalk-color-below-ring	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	stalk-root	0.45	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	stalk-surface-above-ring	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	stalk-surface-below-ring	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	veil-color	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
nursery	children	0.74	NON-ORDINAL	ORDINAL	ORDINAL
	class	0.78	ORDINAL	ORDINAL	ORDINAL
	form	0.20	NON-ORDINAL	NON-ORDINAL	ORDINAL
	has-nurs	0.70	NON-ORDINAL	ORDINAL	ORDINAL
	health	0.50	NON-ORDINAL	ORDINAL	ORDINAL
	housing	0.50	NON-ORDINAL	ORDINAL	ORDINAL
	parents	0.00	NON-ORDINAL	ORDINAL	ORDINAL
	social	0.50	NON-ORDINAL	ORDINAL	ORDINAL
postoperative	BP-STBL	-	NON-ORDINAL	ORDINAL	ORDINAL
	CORE-STBL	-	NON-ORDINAL	ORDINAL	ORDINAL
	L-BP	1.00	ORDINAL	ORDINAL	ORDINAL
	L-CORE	1.00	ORDINAL	ORDINAL	ORDINAL
	L-O2	0.63	NON-ORDINAL	ORDINAL	ORDINAL
	L-SURF	1.00	ORDINAL	ORDINAL	ORDINAL
	SURF-STBL	-	NON-ORDINAL	ORDINAL	ORDINAL
soybean	area-damaged	0.74	NON-ORDINAL	ORDINAL	NON-ORDINAL
	canker-lesion	0.32	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	crop-hist	0.32	NON-ORDINAL	ORDINAL	ORDINAL
	date	0.73	NON-ORDINAL	ORDINAL	ORDINAL
	fruit spots	0.11	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	fruit-pods	0.40	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	germination	0.87	ORDINAL	ORDINAL	ORDINAL
	leafspot-size	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	leafspots-halo	0.87	ORDINAL	ORDINAL	NON-ORDINAL
	leafspots-marg	-	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	mycelium	1.00	ORDINAL	ORDINAL	NON-ORDINAL
	precip	-	NON-ORDINAL	ORDINAL	ORDINAL
	roots	1.00	ORDINAL	ORDINAL	NON-ORDINAL
	seed-tmt	0.50	NON-ORDINAL	NON-ORDINAL	NON-ORDINAL
	severity	1.00	ORDINAL	ORDINAL	ORDINAL
	stem-cankers	0.80	NON-ORDINAL	ORDINAL	NON-ORDINAL
	temp	-	NON-ORDINAL	ORDINAL	ORDINAL

take the outlier detection (boxplot whiskers) as ground truth (when it may be wrong for many datasets). It is not only that these methods ignore semantics, but also that different outlier detection methods may return different sets of outliers depending on the approach they implement (distance and density of data points, statistical models

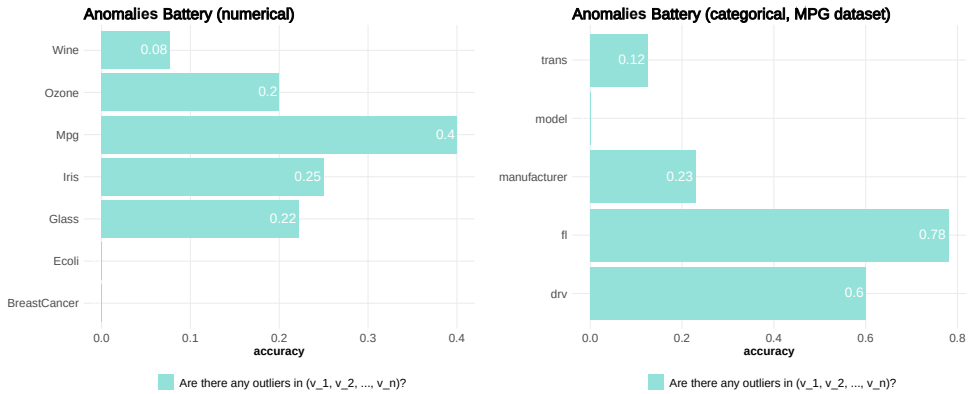

Fig. 6. Average results for the Anomalies Battery for different datasets using numerical features (left), and categorical ones for Mpg dataset (right).

to predict the probability of a dataset distribution, etc.). On the other hand, the performance of GPT-3 is sometimes erratic, obtaining as answers the same set of values it takes as input, possibly indicating that GPT-3 has not correctly understood the task to be performed.

Analysing now the categorical attributes, Figure 6 (right) shows the average results. In this case, although the average accuracy results are somewhat better than in the previous case (0.35), if we analyse the attributes individually, we can observe that GPT-3 is not able to correctly use their semantic information to detect the anomalies. Some examples of this are shown in Table 15 in the appendix. It seems that when we insert anomalies to the attributes with a low number of unique values, anomalies, GPT-3 is able to detect them (see, e.g., attributes "drv" and "fl" in Table 15). However, for more complex attributes, such as the car "model", which includes a many unique alphanumeric terms of different lengths, when trying to detect the introduced anomalies, GPT-3 performs very poorly. In general, from this and the previous experiments we have seen that GPT-3 works well for simple examples of anomaly detection where the context is clear (e.g., {`flat, house, apartment, dinosaur`} or {`70ºF, 71ºF, 71ºF, 110ºF, 71ºF`}). However, for real datasets, it is (still) much more difficult to obtain acceptable results.

Finally, in Figure 7 we show the experimental results for the Imputation Battery. Figure 7 (left) shows the results for those databases representing real cases of specific domains (which we call "domain databases"), namely, the UCs-Satellite, PAF-Address and ATP Players datasets. 'Reg' stands for 'regression' imputation (the output is a numerical value) and 'Class' stands for 'classification' imputation (the output is a a nominal or a textual value). Here, we focus on two learning strategies: zero-shot and 9-shot. We employ the *Full prompt* described in section 3.2. We discard the use of traditional imputation methods for comparison since some of the attributes are textual and they cannot be directly processed by the *DecisionTreeClassifer*. If we analyse the performance for GPT-3, we see that, in general, the results are positive, with 0.48 and 0.80 as average performance for, respectively, zero-shot and 9-shot strategies. Focusing on the 9-shot, the exception to the good results is the dataset

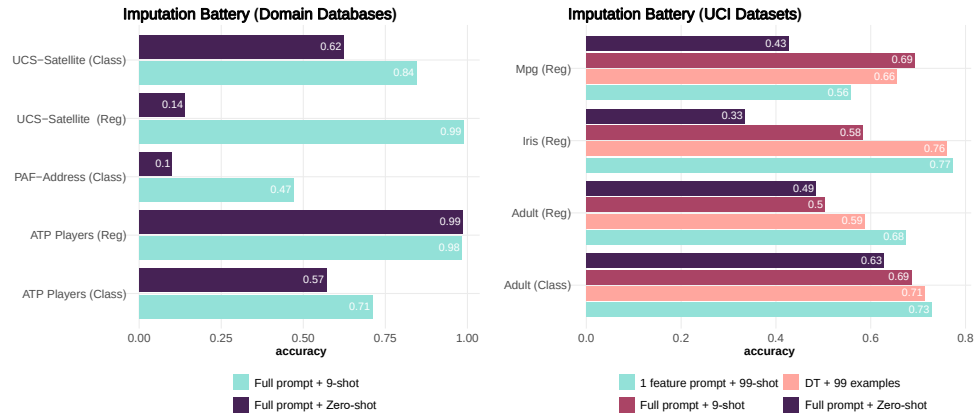

Fig. 7. Average results for the Imputation Battery for domain databases (left), and UCI datasets (right). Dataset results are split by numerical features (Reg) and categorical attributes (Class) when they have features of both types. For the those databases representing specific domains ("domain databases"), we employ the *Full prompt* with 9-shot and zero-shot configurations. For the UCI datasets, we show the performance of the imputation with the *Full prompt* with 9-shot and zero-shot configurations, the *1-Feature prompt* with 99-shot and finally a decision tree trained with 99 examples.

PAF-Address. In this case, the instances are formed by different components of actual addresses in the UK and, for most of the features, the model was not able to correctly infer the individual values of one feature given the others. In some cases language models following a zero-shot learning strategy failed because they were not able to properly identify the task to be performed.

Figure 7 (right) shows the results for the UCI datasets (Mpg, Iris and Adult). Here, we analyse four different strategies: *1-Feature prompt* with 99 examples; *Full prompt* with 9-shot; *Full prompt* with zero-shot; and a *DecisionTreeClassifer* trained with 99 examples and only one feature (the most relevant one with respect to the target). The results in Figure 7(right) show that language models have a comparable effectiveness to imputation methods based on simple predictive models (decision trees). This is specially the case when following the *1-Feature prompt* strategy with 99-shot, which seems to be the best option in all datasets except for MPG.

5 Conclusions

Large language models based on transformers and trained on enormous datasets have recently disrupted artificial intelligence thanks to an unexpected abstraction capacity that has expanded their applicability to fields and problems not originally anticipated. In this work we have analysed different configurations and prompts, as well as the effect of the number of examples (from zero-shot to 99-shot, depending on the problem) to see their performance for a wide range of data wrangling problems. To our knowledge, this paper is the first one that explores the possibilities of language models for data wrangling problems. The results show the capacity of these systems

to learn transformation functions from few examples, to detect data types and domains, determine when there is an order in the attributes and in many cases give the order as well, complete missing data based on semantic components, rather than statistical properties and, to some extent, detect anomalous data. The performance of the studied language models is comparable to well-known systems specialised in data wrangling for some of the batteries, and a good complement that fills new niches for others. These results open a promising research direction to explore the possible applications of language models when used freely through their Application Programming Interfaces (APIs) or when integrated into specialised tools for data wrangling. This is not limited to data wrangling, but could well be used for other tasks in data science, especially those that can be learnt from very few examples and require extensive domain knowledge [15].

Even so, access to large language models is still limited because of cost, infrastructure, privacy issues or lack of training. It is infeasible to use them locally and therefore different subscription models are provided to users via APIs, but this access is still restricted or expensive. We think that this paper comes at the right time, as new open initiatives are emerging to universalise the use of such systems. A notable example is the BigScience[§§§] consortium consisting of 900 researchers from 60 countries and more than 250 institutions. They are jointly creating very large multilingual language models for universal and free access by the scientific community and other professional users. This will make the scenario and pipelines we are describing more common.

All in all, throughout the paper we have tried to understand how we can include language models in the data processing and analysis pipeline. Note that it was not our goal to see for each and every task whether current language models are better than those systems that are specialised in solving data wrangling tasks. Our paper mostly focuses on a general assessment of the possibilities, the range of tasks and prompts data scientists should use for particular cases in their data processing pipelines. The integration of each particular solution into specific tools that maximise performance would end up with a large number of systems the user would need to know and the effort of realising which one serves each particular problem.

Coping with variability with a general and flexible approach instead comes at the cost of some familiarity and maturity of the users. Part of this will come from experience, as they start using language models successfully. For instance, in Table 10 we show the data-wrangling tasks where GPT-3 can help automate according to our experiments, under which conditions the systems are most and least useful. While this only applies for the tasks and GPT-3, it is also possible to set some basic guidelines that can be followed for the general use of language models in data wrangling and other data processing pipelines (see Table 16 in the Appendix).

As future work, we would like to analyse how data wrangling automation can be improved by giving more information to the user about the reliability of the results given by the language models, using their probabilities and determining cutoffs. Some assistance for choosing examples or prompts could also be useful. It is also necessary that other researchers, especially those in human-computer interaction, perform studies with real data scientists and machine learning practitioners using language models for data wrangling. Questionnaires should be conducted to evaluate how effective the

[§§§] `https://bigscience.huggingface.co/`

automation or assistance is. This is necessary to determine how useful these systems are, since the way they are used is very different from other data wrangling systems, and a comparison solely based on performance –ignoring many other factors– will always be partial.

Acknowledgements We thank Lidia Contreras for her help with the Data Wrangling Dataset Repository. We thank the anonymous reviewers from ECMLPKDD Workshop on Automating Data Science (ADS2021) and the anonymous reviewers of this special issue for their comments.

Author contributions All authors contributed equally to this work.

Funding This work was funded by the Future of Life Institute, FLI, under grant RFP2-152, the MIT-Spain - INDITEX Sustainability Seed Fund under project COST-OMIZE, the EU (FEDER) and Spanish MINECO under RTI2018-094403-B-C32, Generalitat Valenciana under PROMETEO/2019/098 and INNEST/2021/317, EU's Horizon 2020 research and innovation programme under grant agreement No. 952215 (TAILOR) and US DARPA HR00112120007 ReCOG-AI.

Availability of data and material The Manipulation Battery is publicly available at `https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/mult_data_wrangling` and `https://github.com/gonzalojaimovitch/lm-dw`.

Code availability All the code and results can be found in `https://github.com/gonzalojaimovitch/lm-dw`.

Declarations

Conflicts of interest No conflicts of interest or competing interests.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Ashok, P., Nawaz, G.K.: Outlier detection method on uci repository dataset by entropy based rough k-means. Defence Science Journal **66**(2), 113–121 (2016)
2. Bellmann, P., Schwenker, F.: Ordinal classification: Working definition and detection of ordinal structures. IEEE Access **8**, 164380–164391 (2020). https://doi.org/10.1109/ACCESS.2020.3021596

3. Ben-Gal, I.: Outlier detection. In: Data mining and knowledge discovery handbook, pp. 131–146. Springer (2005)
4. Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the dangers of stochastic parrots: Can language models be too big? In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. p. 610–623. FAccT '21 (2021)
5. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. The Journal of Machine Learning Research **3**, 1137–1155 (2003)
6. Bhupatiraju, S., Singh, R., Mohamed, A.r., Kohli, P.: Deep API programmer: Learning to program with APIs. arXiv preprint arXiv:1704.04327 (2017)
7. BIG-bench collaboration: Beyond the imitation game: Measuring and extrapolating the capabilities of language models. arXiv preprint arXiv:2206.04615 (2022), `https://github.com/google/BIG-bench/`
8. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
9. Chen, Y., Dang, X., Peng, H., Bart, H.L.: Outlier detection with the kernelized spatial depth function. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(2), 288–305 (2008)
10. Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J.: Automating common data science matrix transformations. In: ECMLPKDD workshop on Automating Data Science. ECML-PKDD '19 (2019)
11. Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., Ramírez-Quintana, M.J., Katayama, S.: Automated data transformation with inductive programming and dynamic background knowledge. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019. ECML-PKDD '19 (2019)
12. Cropper, A., Tamaddoni, A., Muggleton, S.H.: Meta-interpretive learning of data transformation programs. In: Inductive Logic Programming. pp. 46–59 (2015)
13. Das, K., Schneider, J.: Detecting anomalous records in categorical datasets. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 220–229 (2007)
14. Das, K., Schneider, J., Neill, D.B.: Anomaly pattern detection in categorical datasets. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 169–176 (2008)
15. De Bie, T., De Raedt, L., Hernández-Orallo, J., Hoos, H.H., Smyth, P., Williams, C.K.I.: Automating data science: Prospects and challenges. Communications of the ACM, 65 (3), 76-87 arXiv preprint arXiv:2105.05699 (2022)
16. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
17. Dua, D., Graff, C.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
18. Ellis, K., Gulwani, S.: Learning to learn programs from examples: Going beyond program structure. In: IJCAI. pp. 1638–1645 (2017)
19. Fernando, M.P., Cèsar, F., David, N., José, H.O.: Missing the missing values: The ugly duckling of fairness in machine learning. International Journal of Intelligent Systems **36**(7), 3217–3258 (2021)
20. Ferrari, A., Russo, M.: Introducing Microsoft Power BI. Microsoft Press (2016)
21. Furche, T., Gottlob, G., Libkin, L., Orsi, G., Paton, N.W.: Data wrangling for big data: Challenges and opportunities. In: EDBT. vol. 16, pp. 473–478 (2016)
22. Gao, T., Fisch, A., Chen, D.: Making pre-trained language models better few-shot learners. arXiv preprint arXiv:2012.15723 (2020)
23. García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J.M., Herrera, F.: Big data preprocessing: methods and prospects. Big Data Analytics **1**(1), 1–22 (2016)

24. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: Procs. 38th Principles of Programming Languages. pp. 317–330 (2011)
25. Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U., Zorn, B.: Inductive programming meets the real world. Communications of the ACM **58**(11), 90–99 (2015)
26. Ham, K.: OpenRefine (version 2.5). http://openrefine. org. free, open-source tool for cleaning and transforming data. Journal of the Medical Library Association: JMLA **101**(3), 233 (2013)
27. He, Z., Xu, X., Huang, Z.J., Deng, S.: Fp-outlier: Frequent pattern based outlier detection. Computer Science and Information Systems **2**(1), 103–118 (2005)
28. Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., Steinhardt, J.: Measuring massive multitask language understanding. ICLR (2021)
29. Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., Steinhardt, J.: Measuring mathematical problem solving with the MATH dataset. CoRR **abs/2103.03874** (2021), `https://arxiv.org/abs/2103.03874`
30. Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç., Hidalgo, C.: Sherlock: A deep learning approach to semantic data type detection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1500–1508 (2019)
31. Izacard, G., Grave, E.: Leveraging passage retrieval with generative models for open domain question answering. arXiv preprint arXiv:2007.01282 (2020)
32. Jaimovitch-Lopez, G., Ferri, C., Hernandez-Orallo, J., Martinez-Plumed, F., Ramirez-Quintana, M.J.: Can language models automate data wrangling? ECML/PKDD Workshop on Automated Data Science (ADS2021), `https://sites.google.com/view/autods` (2021)
33. Kandel, S., Paepcke, A., Hellerstein, J., Heer, J.: Wrangler: Interactive visual specification of data transformation scripts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 3363–3372. ACM (2011)
34. Lazarevic, A., Kumar, V.: Feature bagging for outlier detection. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 157–166 (2005)
35. Lu, Y., Bartolo, M., Moore, A., Riedel, S., Stenetorp, P.: Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. arXiv preprint arXiv:2104.08786 (2021)
36. Nazabal, A., Williams, C.K., Colavizza, G., Smith, C.R., Williams, A.: Data engineering for data analytics: a classification of the issues, and case studies. arXiv preprint arXiv:2004.12929 (2020)
37. Noto, K., Brodley, C., Slonim, D.: Frac: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. Data mining and knowledge discovery **25**(1), 109–133 (2012)
38. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
39. Petrova-Antonova, D., Tancheva, R.: Data Cleaning: A Case Study with OpenRefine and Trifacta Wrangler. In: International Conference on the Quality of Information and Communications Technology. pp. 32–40. Springer (2020)
40. Porwal, U., Mukund, S.: Outlier detection by consistent data selection method. arXiv preprint arXiv:1712.04129 (2017)
41. Puri, R., Catanzaro, B.: Zero-shot text classification with generative language models. arXiv preprint arXiv:1912.10165 (2019)
42. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)

43. Raman, V., Hellerstein, J.M.: Potter's wheel: An interactive data cleaning system. In: VLDB. vol. 1, pp. 381–390 (2001)
44. Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S.G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J.T., et al.: A generalist agent. arXiv preprint arXiv:2205.06175 (2022)
45. Rubin, D.B.: Inference and missing data. Biometrika **63**(3), 581–592 (1976)
46. Schick, T., Schütze, H.: Exploiting cloze questions for few-shot text classification and natural language inference. arXiv preprint arXiv:2001.07676 (2020)
47. Shannon, C.E.: Communication theory of secrecy systems. The Bell system technical journal **28**(4), 656–715 (1949)
48. Shi, Y., Li, W., Sha, F.: Metric learning for ordinal data. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)
49. Singh, R., Gulwani, S.: Predicting a correct program in programming by example. In: International Conference on Computer Aided Verification. pp. 398–414. Springer (2015)
50. Singh, R., Gulwani, S.: Transforming spreadsheet data types using examples. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 343–356 (2016)
51. Sleeper, R.: Tableau Desktop Pocket Reference. " O'Reilly Media, Inc." (2021)
52. Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al.: Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. arXiv preprint arXiv:2201.11990 (2022)
53. Tamkin, A., Brundage, M., Clark, J., Ganguli, D.: Understanding the capabilities, limitations, and societal impact of large language models. arXiv preprint arXiv:2102.02503 (2021)
54. Terrizzano, I.G., Schwarz, P.M., Roth, M., Colino, J.E.: Data Wrangling: The Challenging Journey from the Wild to the Lake. In: CIDR (2015)
55. Tools, D.W.: Software— trifacta. URL https://www.trifacta.com
56. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)
57. Wei, J., Bosma, M.P., Zhao, V., Guu, K., Yu, A.W., Lester, B., Du, N., Dai, A.M., Le, Q.V.: Finetuned language models are zero-shot learners (2022), `https://openreview.net/forum?id=gEZrGCozdqR`
58. Wu, B., Szekely, P., Knoblock, C.A.: Learning data transformation rules through examples: Preliminary results. In: Information Integration on the Web. p. 8 (2012)
59. Xu, S., Semnani, S.J., Campagna, G., Lam, M.S.: AutoQA: From databases to QA semantic parsers with only synthetic training data. EMNLP (2020)
60. Zeng, W., Ren, X., Su, T., Wang, H., Liao, Y., Wang, Z., Jiang, X., Yang, Z., Wang, K., Zhang, X., et al.: Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. arXiv preprint arXiv:2104.12369 (2021)
61. Zhang, D., Suhara, Y., Li, J., Hulsebos, M., Demiralp, Ç., Tan, W.C.: Sato: Contextual semantic type detection in tables. arXiv preprint arXiv:1911.06311 (2019)
62. Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., Fedus, W.: Designing effective sparse expert models. arXiv preprint arXiv:2202.08906 (2022)

Supplementary Material

Table 10. Illustrative (simple and natural) prompts explored to semi-automate all the data wrangling batteries. When different alternatives are provided, those with the best results in our preliminary experiments are chosen for the rest of the experiments in the paper. The last two columns provide some tips on the application of GPT3 for the automation of different data wrangling tasks.

Battery	Prompt Template		Notes	When does it work well?	When does it work badly?
	Examples (n-shot)	Question			
Manipulation	Input: 'X'\nOutput: 'Y'\n\n	Input: 'Z'\nOutput:	No alternatives. We wanted a generalisation from the concept itself and we tested a varying number of n-shots. The semantic context about the domain is clear.	Tasks that can be solved by simple string transformation	Reasoning or calculation capabilities are required (e.g., identification of dimensions, coefficients, arithmetic operations, etc.). Domain information required (e.g., date format)
Types		What is the best domain that describes the values in $\{v_1, v_2, \ldots, v_n\}$?		Asking for the 'semantic type' of the values. Working with nominal variables	Asking for the 'domain' of the values. Working with numerical variables
		What is the best semantic type that describes the values in $\{v_1, v_2, \ldots, v_n\}$?			
		What type of data is $\{v_1, v_2, \ldots, v_n\}$?	Alternative. Not very specific. Worst results in the initial tests		
		What domain of data is $\{v_1, v_2, \ldots, v_n\}$?	Alternative. Not very specific. Worst results in the initial tests		
		What are $\{v_1, v_2, \ldots, v_n\}$?	Alternative. Not very specific. Worst results in the initial tests		
		What is the domain of the set $\{v_1, v_2, \ldots, v_n\}$?	Alternative. Not very specific. Worst results in the initial tests		
Ordinal	Is 'house' higher than 'apartment'? Yes Is 'apartment' higher than 'house'? No Is 'red' higher than 'blue'? No Is 'blue' higher than 'red'? No Is 'old' higher than 'young'? Yes Is 'young' higher than 'old'? No Is 'totally agree' higher than 'agree'? Yes Is 'agree' higher than 'totally agree'? No Is 'New York' higher than 'Chicago'? No Is 'Chicago' higher than 'New York'? No Is 'Heavy rain' higher than 'Showers'? Yes Is 'Showers' higher than 'Heavy rain'? No	Is "v_1" higher than "v_2"?	The context is always the same for all the examples (12-shot), while only the 13th line changes. This long context is added because it helps to frame the question and gives better results. We tried with some other terms different from 'higher', such as 'larger', 'more', etc., but we got worst results in the initial tests for these variations	Telling ordinal vs non-ordinal. Working with attributes that have a textual representation of numbers or intervals (e.g., age, tumor-size, etc.).	Recognising the actual order of complex attributes or dealing with time if the adjective is not adjusted. Dealing with absent values
	Is (low, medium, high) an ordinal? Yes Is (door, window, wheel) an ordinal? No	Is $\{v_1, v_2, \ldots, v_n\}$ an ordinal?	Simpler version: 2-shot (always the same) and the question.		
Anomalies		Are there any outliers in $\{v_1, v_2, \ldots, v_n\}$?		Dealing with anomalies of attributes with a low number of unique values	Dealing with attributes with many unique alphanumeric terms of different lengths
		Which is the odd-one-out in $\{v_1, v_2, \ldots, v_n\}$?	Alternative. Despite being equivalent to 'oulier', the initial tests showed worse results.		
Imputation	feature_1: v_1, feature_2: v_2, ..., feature_n: v_3 feature_1: v_4, feature_2: v_5, ..., feature_n: v_6 ... feature_1: v_7, feature_2: v_8, ..., feature_n: v_9	feature_1: v_n, feature_2: v_m, ..., feature_n:	Full prompt strategy using all the available features in the dataset and different n-shot strategies	Relevant attributes are provided. Imputation for both categorical and numerical data	Complex attributes (e.g., addresses)
	feature_x: v_1, feature_y: v_2 ... feature_z: v_3, feature_y: v_4	feature_x: v_5, feature_y:	1-Feature prompt using only the most relevant feature and different n-shot strategies		

Fig. 8. Overview of results obtained by the different versions of GPT-3 models disaggregated by learning strategy (from zero-shot to 10-shot learning), system, domain and data wrangling task. Tasks per domain sorted by average performance.

Table 11. Wrong answers provided by the GPT-3 DaVinci system setting (examples based on large sentences are represented in Table 12). 4-shot setting used for illustrative purposes.

Input 1	Input 2	Input 3	Output	Correct output	Problem	Domain
1990.18.01	-	-	01.18.1990	01/18/1990	changeFormat-1	dates
28-10-2001	-	-	10/28/2001	08/23/2010	changeFormat-2	dates
99/09/03	-	-	09/99/03	09/03/99	changeFormat-3	dates
06-04-2001	-	-	04/06/2001	06/04/2001	changeFormat-3	dates
08-07-2017	-	-	07-08-2017	08-07-2017	changeFormat-4	dates
31/05/17	-	-	17	31	getDay-1	dates
25-08-85	-	-	08	25	getDay-1	dates
31/03/75	-	-	31st	31th	getDayOrdinal-1	dates
09/11/53	-	-	11th	09th	getDayOrdinal-1	dates
05/09/2008	-	-	5th	05th	getDayOrdinal-2	dates
Thursday, October 31, 1985	-	-	October	Thursday	getMonthName-1	dates
11/06/2016	-	-	Jun	Nov	reduceMonthName-2	dates
8072017	-	-	07-08-2008	08-07-2017	setFormat-1	dates
2261993	-	-	93-22-1993	26-02-1993	setFormat-1	dates
8222000	-	-	22/02/2000	12/08/2000	setFormat-2	dates
11202001	-	-	20/12/2001	10/11/2001	setFormat-2	dates
casper	canal13	-	casper@canal13.cl	casper@canal13.com	generate-3	emails
caedgigo@garmendia.cl	-	-	gmail.com	garmendia.cl	getAfterAt-2	emails
1-845-456-7891	-	-	18454567819	18454567891	deletePunctuation-1	freetext
1-845-333-7891	-	-	18453337890	18453337891	deletePunctuation-1	freetext
1-7891	-	-	7891	17891	deletePunctuation-2	freetext
1-7892	-	-	7892	17892	deletePunctuation-2	freetext
Aliquam	-	-	a	A	firstCharacter-2	freetext
Computer and Communications Security	-	-	ACSAC	CCS	getCaps-2	freetext
Jones	1	-	Mr. Jones	Sr. Jones	addTitle-2	names
Dario Gag-Dorado	-	-	dari	daga	login-1	names
Paco Gabot Narale	-	-	paco	pagana	login-1	names
Daman Hivser-Kleiner	-	-	dakle	dahi	login-2	names
Giussepe Hindravtoks	-	-	giussepe	gihi	login-2	names
Agustino	Zimmann	-	Agustino, A.	Agustino, Z.	reduceName-5	names
618-4390	PAN	-	(7) 618-4390	(507) 618-4390	countryPrefix-3	phones
36-678-59-10	AUT	-	(9) 36-678-59-10	(43) 36-678-59-10	countryPrefix-7	phones
846-2730	AND	-	(846) 2730	(376) 846-2730	countryPrefix-7	phones
425-846-2730 425-425 425	-	-	425-425 425	425-846-2730	getNumber-2	phones
618 4390	425	-	425-618-4390	725-618-4390	setPrefix-1	phones
743-1650	425	-	425-743-1650	892-743-1650	setPrefix-1	phones
618-4390	425	-	(+425) 618-4390	(+725) 618-4390	setPrefix-5	phones
743-1650	425	-	(+425) 743-1650	(+892) 743-1650	setPrefix-5	phones
08	5	-	12	13	addTime-1	times
16:15:12	5	-	05:15:12	21:15:12	addTime-1	times
21:20	5	-	26:20	02:20	addTime-2	times
16:15:12	-	-	16:15:12:00	16:15:12	appendTime-1	times
16:15:12	30	-	16:15:12:30	16:15:12	appendTime-3	times
06:15:12	-	-	06:15:12	16:15:12	convert-1	times
08:40	EST	PST	20:40:00	05:40:00	convert-10	times
06:15	CET	MST	00:15:00	22:15:00	convert-10	times
14:10	12h	-	14:10	02:10 PM	convert-3	times
21:20	12h	-	21:20	09:20 PM	convert-3	times
08:40 UTC	24h	-	20:40	08:40	convert-4	times
06:15:12	24h	-	06:15:12	16:15:12	convert-4	times
14:10	-	-	14:10 AM	02:10 PM	convert-5	times
21:20	-	-	21:20 AM	09:20 PM	convert-5	times
08:40 UTC	-	-	03:40 PM	08:40 AM	convert-6	times
14:10	CET	UTC	14:10	13:10	convert-7	times
21:20	EST	UTC	03:20	02:20	convert-7	times
08:40	EST	PST	20:40	05:40	convert-8	times
06:15:12	CET	MST	18:15:12	22:15:12	convert-8	times
14:10	CET	UTC	14:10:00	13:10:00	convert-9	times
21:20	EST	UTC	07:20:00	02:20:00	convert-9	times
08	-	-	08	-	deleteTime-1	times
12.20dg	mg	-	12.200001	1220	convert-1	units
1854 dam	dm	-	18.540000	185400	convert-1	units
92.26 K	cK	-	92.26	9226	convert-2	units
12.2	dg	mg	12.200001	1220	convert-3	units
1854	dam	dm	18.540000	185400	convert-3	units
92.26	K	cK	92.26	9226	convert-4	units
81	hm	cm	0.081	8100	convert-4	units
1854 dam	-	-	Area	Length	getSystem-1	units
12.20dg	-	-	12.20	12.2	getValue-1	units

Table 12. Wrong answers provided by the GPT-3 DaVinci system for examples from the Manipulation Battery based in large sentences. 4-shot setting used for illustrative purposes.

		problem	**domain**
Input	Etiam dapibus tortor sed cursus blandit.		
Output	Etiamdapibustortorsedcursussed	deletePunctuation-3	freetext
Correct output	Etiamdapibustortorsedcursusblandit		
Input	Quisque mattis dolor ac metus fermentum consectetur.		
Output	Quismattisdoloracmetusfermentumconsectetur	deletePunctuation-3	freetext
Correct output	Quisquemattisdoloracmetusfermentumconsectetur		
Input	Integer malesuada elit et hendrerit fermentum.		
Output	Integermalesuadaelithendreritfermentum	deletePunctuation-4	freetext
Correct output	Integermalesuadaelitethendreritfermentum		
Input	Pellentesque tempor justo a dui varius maximus.		
Output	Pellentesque temporjustoaduivariusmaximus	deletePunctuation-4	freetext
Correct output	Pellentesquetemporjustoaduivariusmaximus		
Input	Mauris fringilla neque nec libero consequat quis facilisis nisi tristique		
Output	Maurisfringillanequeconsequatlibero	deleteSpaces-2	freetext
Correct output	Maurisfringillanequenecliberoconsequatquisfacilisisnisitristique		
Input	Nam cursus purus nec mi maximus fermentum		
Output	Namcursuspurusnecminimaximusfermentum	deleteSpaces-2	freetext
Correct output	Namcursuspurusnecmimaximusfermentum		
Input	Quisque mattis dolor ac metus fermentum consectetur.		
Output	QUISQUE MATTIS DOLOR AC METUS FERMENTUM CONSEQUETUR	toUpper-1	freetext
Correct output	QUISQUE MATTIS DOLOR AC METUS FERMENTUM CONSECTETUR		

Table 13. Illustrative examples of answers provided by the GPT-3 DaVinci system for some columns from the Types Battery.

Zillow	**Prompt1**
Input	What is the best domain that describes the values in {Palmyra, Defiance,. . . ,Verona}?
Output	The best domain that describes the values in {Palmyra, Defiance,. . . ,Verona} is ”small towns”
Correct type	city
Input	What is the best domain that describes the values in {Norman, Frackville, . . . , Corte Madera}?
Output	The best domain would be ”towns in the United States.”
Correct type	city
	Prompt2
Input	What is the best semantic type that describes the values in {Palmyra, Defiance,. . . ,Verona}?
Output	The best semantic type that describes the values in {Palmyra, Defiance,. . . ,Verona} is Place
Correct type	City
Input	What is the best semantic type that describes the values in {Norman, Frackville, . . . , Corte Madera}?
Output	The best semantic type that describes the values in the given set is ”cities.”
Correct type	City
MedicalNoShow	**Prompt1**
Input	What is the best domain that describes the values in {F, M, M, F, M, M, M, M, M, M}?
Output	The best domain would be ”Frequency of Males.”
Correct type	Gender
Input	What is the best domain that describes the values in {F, M, M, F, F, M, F, M, F, M}?
Output	The best domain would be ”Females and Males.”
Correct type	Gender
	Prompt2
Input	What is the best semantic type that describes the values in {F, M, M, F, M, M, M, M, M, M}?
Output	The best semantic type that describes the values in {F, M, M, F, M, M, M, M, M, M} is ”Gender.”
Correct type	Gender
Input	What is the best semantic type that describes the values in {F, M, M, F, F, M, F, M, F, M}?
Output	The best semantic type that describes the values in {F, M, M, F, F, M, F, M, F, M} is ”Gender.”
Correct type	Gender
SpeedDating	**Prompt1**
Input	What is the best domain that describes the values in {28, 20, 32, 28, 42, 34, 37, 34, 28, 30}?
Output	The best domain would be 28-42.
Correct type	Age
Input	What is the best domain that describes the values in {29, 55, 23, 55, 28, 21, 31, 33, 29, 34}?
Output	The best domain would be {21, 23, 28, 29, 31, 33, 34, 55}.
Correct type	Age
	Prompt2
Input	What is the best semantic type that describes the values in {28, 20, 32, 28, 42, 34, 37, 34, 28, 30}?
Output	The best semantic type that describes the values in {28, 20, 32, 28, 42, 34, 37, 34, 28, 30} is a set of whole numbers.
Correct type	Age
Input	What is the best semantic type that describes the values in {29, 55, 23, 55, 28, 21, 31, 33, 29, 34}?
Output	The best semantic type that describes the values in {29, 55, 23, 55, 28, 21, 31, 33, 29, 34} is a set of whole numbers.
Correct type	Age

Table 14. Illustrative examples of answers provided by the GPT-3 DaVinci system for numerical attributes from the Anomalies Battery.

Dataset (Attribute)		
BreastCancer (chromatin)	**Input**	Are there any outliers in {1, 3, 4, 1, 2, 6, 4 . . . }?
	Output	The outliers are 9, and 10.0.
	Boxplot Output	The outlier is 10
Ecoli (mcg)	**Input**	Are there any outliers in {0.0, 0.04, 0.06, 0.1, 0.11, 0.12, 0.16, 0.17, 0.18, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, . . . }?
	Output	The outliers are 0.0, 0.04, 0.06, 0.1, 0.11, 0.12, 0.16, 0.17, 0.18, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25, . . .
	Boxplot Output	There are no outliers.
Wine (mcg)	**Input**	Are there any outliers in {3.06, 0.76, 3.24, 1.49, 4.69, 3.39, . . . }?
	Output	The outliers are 0.47, 0.61, 0.63, and 0.66. . . .
	Boxplot Output	There are no outliers.
Glass (Na)	**Input**	Are there any outliers in {10.73, 11.2, 14.3, 12.07, 13.3, . . . }?
	Output	The outliers are 10.73, 11.02, 11.03, 11.23, and 15.79.
	Boxplot Output	The outliers are 10.73, 11.02, 11.03, 11.23, 11.45, 15.79, 17.38
Iris (Sepal.Width)	**Input**	Are there any outliers in {3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9, . . . }?
	Output	The outliers are 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2 and 4.4.
	Boxplot Output	There are no outliers.
Mpg (hwy)	**Input**	Are there any outliers in {29, 29, 31, 30, 26, 15, 20, 17, 26, . . . }?
	Output	The outliers are 5, 12, 14, 18, and 17.
	Boxplot Output	The outliers are 44, 44, 41.

Table 15. Illustrative examples of correct and wrong answers provided by the GPT-3 DaVinci system for categorical attributes from the Anomalies Battery. Inserted synthetic anomalies highlighted in yellow.

Attribute		
model	**Input**	Are there any outliers in {a4, a4 quattro, a6 quattro, c1500 suburban 2wd, corvette, k1500 tahoe 4wd, malibu, caravan 2wd, dakota pickup 4wd, durango 4wd, ram 1500 pickup 4wd, expedition 2wd, **1.8**, explorer 4wd, f150 pickup 4wd, mustang, civic, sonata, tiburon, grand cherokee 4wd, range rover, navigator 2wd, mountaineer 4wd, altima, maxima, pathfinder 4wd, grand prix, forester awd, impreza awd, 4runner 4wd, camry, camry solara, corolla, land cruiser wagon 4wd, toyota tacoma 4wd, gti, jetta, new beetle, passat, . . . }?
	Output	There are no outliers in this data
	Correct Output	The outlier is 1.8
trans	**Input**	Are there any outliers in {auto(l5), manual(m5), manual(m6), auto(av), auto(s6), auto(l4), auto(l3), auto(l6), **honda**, auto(s5), auto(s4), . . . }?
	Output	There are no outliers in this data
	Correct Output	The outlier is honda
model	**Input**	Are there any outliers in {audi, chevrolet, dodge, ford, honda, hyundai, jeep, land rover, lincoln, mercury, nissan, **r**, pontiac, subaru, toyota, volkswagen, . . . }?
	Output	There are no outliers in this data
	Correct Output	The outlier is r
manufacturer	**Input**	Are there any outliers in {audi, chevrolet, dodge, ford, honda, hyundai, **f**, jeep, . . . }?
	Output	There are no outliers in this data
	Correct Output	The outlier is f
fl	**Input**	Are there any outliers in {p, **24**, r, e, d, c, . . . }?
	Output	The outlier is 24
	Correct Output	The outlier is 24
drv	**Input**	Are there any outliers in {"f" "4" "r", **auto(l5)** . . . }?
	Output	The outlier is auto(l5)
	Correct Output	The outlier is auto(l5)

Table 16. Generic instructions providing a basic skeleton for the incorporation of language models in the automation or assistance of this type of tasks. They should be customised depending on the problem to be solved, the capabilities required for its resolution, the ability of the user as a good prompter, the requirements and costs of the language models, and the evaluation criteria used to determine the efficiency of the whole pipeline.

Guidelines

1. Familiarise with language models using the interactive interface but also programmatically through an API. For instance, replicating the results for some of the batteries in this paper could be a good training for users.
2. Determine and understand the availability, quotas and any restriction about the language models for which regular access is available.
3. Identify the data wrangling problem, and, if possible, associate it with the taxonomy in Table 1.
4. Identify how the problem appears in the data processing pipeline, if it is a one-off process or it is going to be repetitive. This will quantify the scaling factor and the potential gain of automation.
5. Determine if language models are a good fit: if the task requires semantic content, only a few examples and simple reasoning, then language models may be an option. Always think of other better or cheaper alternatives.
6. Derive and try some prompts for the task at hand. These can be few-shot input-output prompts using several examples, zero-shot prompts describing the task to do for a single input, or a combination of both.
7. Decide the strategy to extract the results from the language model automatically and how their quality is going to be evaluated.
8. Run some preliminary tests with some samples, variations of prompts and extraction processes. Derive the quality metrics, the total human time invested and the number of tokens/compute used by the system.
9. If the result is sufficiently effective considering all the factors derived in the point above, then run the data wrangling process with all the data.
10. Document the process and the lessons-learned for the future, and disseminate the experience with other colleagues inside and beyond the organisation that regularly face data wrangling problems.