AVATAR — Automated Feature Wrangling for Machine Learning

Gust Verbruggen, Elia Van Wolputte, Sebastijan Dumančić, and Luc De Raedt

KU Leuven, Department of Computer Science, Leuven, Belgium
Leuven.AI — KU Leuven Institute for AI, Leuven, Belgium
firstname.lastname@kuleuven.be

Abstract. A large part of the time invested in data science is spent on manual preparation of data. Transforming wrongly formatted columns into useful features takes up a significant part of this time. We present the AVATAR algorithm for automatically learning programs that perform this type of *feature wrangling*. Instead of relying on users to guide the wrangling process, AVATAR directly uses the predictive performance of machine learning models to measure its progress during wrangling. We use datasets from Kaggle to show that AVATAR improves raw data for prediction, and square it off against human data scientists.

Keywords: data wrangling, program synthesis, machine learning

1 Introduction

Data scientists spend a lot of time simply preparing data for analysis. Even before exploratory analysis, data cleaning and feature engineering, an additional step of *data wrangling* is often required. This step consists of taking raw data and *wrangling* it into a format that can be used for data science tasks, such as visualisation and prediction. Data wrangling is typically carried out by writing a program that transforms part of the data into the desired format. Writing these programs is very time consuming to data scientists—according to popular statistic, up to 80% of the time in the whole data science pipeline is invested in wrangling [2], which explains the interest in automated data wrangling [10, 5].

The existing work on automated data wrangling, however, assumes that a user knows which format the data should take and can provide input. This input can take many forms. Early methods interactively propose transformations based on data that the user selects and require the wrangling algorithm to learn how to extract the selection [14, 7]. Other methods allow the user to give an example of what the output should look like in order to learn a program that correctly produces this output [5, 3]. This is clearly a strong assumption: the big lesson of feature engineering is that users rarely know which features, and in which form, are useful for the target task.

Taking inspiration from both feature engineering and data wrangling, we introduce the problem of *feature wrangling*, which is concerned with wrangling at the feature level. More specifically, we automatically search for transformations that wrangle individual columns into features of high quality to be used in predictive models. For example, a date formatted as “01/01/2001” would be split in its constituent day, month and year parts and these should be marked as ordinal (day and month) or numerical (year) features. As the resulting features are to be used in supervised machine learning, the quality of the generated features can be assessed using the predictive performance of the resulting model. A major benefit of our approach is that it eliminates the need for user interaction.

Motivational example Consider the excerpt of basketball data in Figure 1a. Regardless of the task, we can see that it is not very suited for further analysis. The **height** feature is not numerical and **position** is ambiguous—is “G-F” a position on its own or is this player comfortable in multiple positions?

Suppose a fourth column **salary** exists that we want to predict. We can then try different possibilities of representations for **position** and **height** and use their performance in predicting **salary** to choose the most appropriate one. For example, **position** can be one-hot encoded or split on “-” and then encoded with a dummy variable for every symbol.

Whereas previous approaches would require the user to provide an example of the desired feature, AVATAR generates and tries different alternatives to see which ones yield the best performance. In this example, it will also discover that splitting **height** by a “-” yields new columns that, after being made numerical in a second iteration, are good features. The full wrangling program and its result are respectively shown in Figures 1b and 1c.

name	**position**	**height**
Kuzma	F	6-9
Wagner	C	6-11
Ingram	G-F	5-11

(a) Initial dataset D

```
splitDummies(position, “-”)
split(height, “-”)
makeNumeric(s1)
makeNumeric(s2)
drop(position)
drop(height)
```

(b) Wrangling program

name	s_3	s_4	s_5	s_1	s_2
Kuzma	1	0	0	6.0	9.0
Wagner	0	1	0	6.0	11.0
Ingram	1	0	1	5.0	11.0

(c) Wrangled dataset D'

Fig. 1: Example of data wrangling for machine learning.

Contributions In this paper, we make the following contributions.

- We introduce the problem of automated feature wrangling, which is concerned with wrangling at the individual feature level and uses the performance of the predictive models to evaluate alternative feature sets.
- We implement this idea in a prototype feature wrangling tool called AVATAR—the Automated VAlue Transformator And extractoR. Given only a dataset and a prediction task, it returns a new dataset with wranlged features that are more suitable for the given task.
- We evaluate AVATAR on real datasets from the Kaggle[1] data science platform.

[1] www.kaggle.com

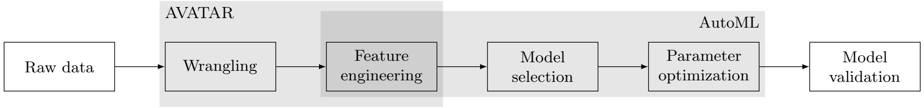


Fig. 2: Overview of the data science pipeline.

2 Related work

Feature engineering aims to improve the performance of predictive models by transforming and combining existing features into new features that are easier for the model to use. Being a laborious process, automating it is an active area of research [8, 9]. The goal of feature wrangling is to use wrangling transformations in order to extract new features from previously unusable columns. Feature wrangling therefore lies at the intersection of data wrangling and feature engineering.

AutoML is concerned with automating the data science pipeline as a whole. The general structure of such a pipeline is shown in Figure 2. These systems start either from the feature engineering or model selection steps and build a data science pipeline with the goal of optimising performance on a prediction task. Some examples of methods are TPOT [13], auto-sklearn [4] and OBOE [18]. Automated feature wrangling can be viewed as extending these approaches to include wrangling transformations in the feature engineering process.

Two types of wrangling approaches aim to prepare data for the data science pipeline. The first is concerned with extracting and restructuring data, as lots of information is still stuck in inconvenient formats such as spreadsheets, XML or json. Selecting a few examples of desired rows allows methods like FlashExtract [10] to learn a program that extracts all similar rows. Auto-Suggest recommends data preparation steps, such as pivot and join, for raw tables [17]. Foofah [6] and AutoPandas [1] learn full transformation programs if an output example can be given. A second type of wrangling is concerned with transforming and normalizing individual columns based on examples provided by a user [5, 3]. Approaches in both types of wrangling assume that a user knows how to represent their data and provide a shortcut to obtain this representation. AVATAR, on the other hand, automatically determines a suitable representation at the feature level.

3 Data wrangling for machine learning

Data wrangling in general is concerned with preparing raw data for data science. In this paper, we focus on the specific task of transforming wrongly formatted columns into usable features by automatically constructing a transformation program—similar to how a human data scientist would perform the same task. We formally describe this problem and present a simple language capable of performing common data wrangling tasks in the following two sections.

3.1 Problem statement

We are interested in learning a data transformation program $P : \mathcal{X} \rightarrow \mathcal{X}'$ that transforms a dataset D with instances of the form $(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y})$ into a new dataset D'. The goal is to obtain a dataset D' with a better feature representation than the original dataset D to perform a given machine learning task. In this paper, we consider the task of supervised learning. The dataset D is used to learn a model $m : \mathcal{X} \rightarrow \mathcal{Y}$ that predicts the value of y given its features $\mathbf{x}$. This model m is learned using a learner $\mathcal{F}$ on the dataset D, written as $m = \mathcal{F}(D)$.

Assessing whether a dataset D' contains better features than D is possible in the existence of a scoring function $s : (\mathcal{F}, D) \rightarrow \mathbb{R}$ that estimates the performance of a model $\mathcal{F}(D)$. The score of a model trained on a dataset then serves as a proxy to evaluate the quality of the features of this dataset—better features will result in better predictions. We assume the learner and scoring function to be given, for example, a decision tree classifier and predictive accuracy.

The problem of feature wrangling for machine learning is then as follows. **Given** a dataset D and a transformation language $\mathcal{L}$, **find** a program $P^* = \arg\max_{P \in \mathcal{L}} s(\mathcal{F}, P(D))$ that transforms D into a dataset $D^* = P^*(D)$ on which an optimal model $\mathcal{F}(D^*)$ can be learned.

It is intractable to find the optimal program, however, as an infinite number of programs can be generated. Any program P such that $s(\mathcal{F}, P(D)) > s(\mathcal{F}, D)$ is an improvement over using the raw data.

3.2 A language for feature wrangling

Let us write $D = [X_1, \ldots, X_m]$ when referring to columns of data. Let $t(X, \mathbf{a})$ be a transformation that takes a column X and (optional) arguments $\mathbf{a}$, and returns a new matrix of columns $\mathbf{X}$. A transformation with fixed arguments is called a *wrangling transformation* and written as $t(\mathbf{a})$. This wrangling transformation is *valid* for X if $t(X, \mathbf{a}) \neq X$.

Example 1. The $\mathsf{split}(X, d)$ transformation takes a column X and a delimiter d. It returns a set of columns obtained by splitting each row of X at every occurrence of d. An example of a wrangling transformation is split(“-”). It is valid for the **height** and **position** columns in Figure 1a, but not for the **name** column.

A wrangling program is simply a sequence of wrangling transformations. Data scientists typically build such programs by iteratively picking a transformation t and arguments $\mathbf{a}$ for a target column X_i such that $t(X_i, \mathbf{a})$ yields new columns. Each of these new columns is given a unique identifier and added to the data.

Example 2. An example of a dataset, wrangling program and its result is shown in Figure 1a. Table 1 shows an overview of transformations currently supported by AVATAR.

Table 1: Overview of transformations supported by AVATAR and their generators. The implicit column parameter is not mentioned. Argument d is a string, p is a regular expression and L is a list of strings .

Transformation	Description	Generator
makeNumerical	Make column numerical.	`true` if X_i contains a number.
oneHot	One hot encode.	`true` if X_i contains limited number of unique values.
NaN	Encode value as hidden missing value.	Values in X_i that match a predefined set of patterns, such as "?" and "111".
split(d)	Split on delimiter.	Strings consisting of subsequent, non-alphanumeric characters found in X_i.
splitDummies(d)	Split on delimiter and dummy encode the resulting categorical features.	Same as `split`.
extractNumber(p)	Extract numbers that follow a regular expression pattern.	Extract numbers from X_i and generate regexes from them by mapping digits to `\d`. Additionally, generate patterns where consecutive `\d` are mapped to `\d+`.
extractWord(L)	Extract a specific word from L in each row.	Greedily look for combinations of words such that each row contains exactly one of these words.
wordToNumber	Convert written numbers to numerical.	`true` if at least one written number is found.

3.3 Generating arguments

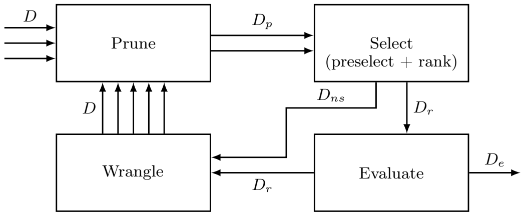
To compose valid wrangling programs, we need to tractably identify the possible arguments of transformation functions. We do so by following a generator-based approach [1]. For each transformation t, we define a generator $\mathcal{G}_t(X)$ that takes a column X as input and yields arguments $\mathbf{a}$ such that $t(\mathbf{a})$ is a valid wrangling transformation for X. In other words, the arguments of wrangling transformations are generated from data and are not predefined by a user. Generators in AVATAR are only allowed to yield a finite number of arguments.

Example 3. The generator for `split` yields strings of consecutive, non-alphanumeric characters from rows in the input column, one at a time. Given either position or height columns from Figure 1a, only a single argument is generated: "-".

Generators for all transformations that AVATAR supports are described in Table 1. A generator for a transformation without additional arguments returns `true` if the function can be applied to X.

4 Machine learning for feature wrangling

At the core of AVATAR is the use of machine learning models for evaluating progress during wrangling. As opposed to looking for a single transformation at a time, multiple transformations are considered in parallel to allow feature interactions to be considered when evaluating progress. In order to do so, AVATAR explores the space of wrangling programs by exploiting the fact that, starting from a dataset, the order in which transformations are applied to this dataset



(a) Graphical overview. Arrows represent columns.

```
def AVATAR($D$):
  while true:
    $D_p$ = prune($D$)
    $D_s, D_{ns}$ = preselect($D_p$)
    $D_r$ = rank($D_s$)
    $D_e$ = evaluate($D_r$)
    if stop():
      return previous $D_e$
    $D$ = wrangle($D_r + D_{ns}$)
```

(b) Algorithmic overview.

Fig. 3: The AVATAR algorithm.

does not matter. At each iteration, a large wrangling program is generated, which is then pruned by subsequent steps. A high level overview of AVATAR is shown in Figure 3 and the following sections describe each of these steps in detail.

4.1 Prune

Pruning aims to remove columns that are not and will never become useful features. This step allows the generators for transformations to be significantly less complex, as different edge cases don't have to be explicitly considered. The following columns are removed from the dataset.

1. Columns that are constant.
2. Columns in which more than p_{nan} percent of values is missing.
3. Columns that are more than p_{id} percent identical to another column.

4.2 Select

From all remaining columns, the selection step aims to find promising features—those that have at least some predictive power. Selection happens in two steps: preselection and feature ranking. This ranking of features is then used in the next step to evaluate the fitness of the current dataset.

Preselection This step heuristically excludes the following bad columns.

1. Categorical columns that contain more than p_u percent unique values.
2. Categorical columns X_j for which there exists a column X_i with $i < j$ such that a bijective mapping exists between these columns for at least p_{bi} of rows.

Preselection serves to reduce the effort required by the feature ranking step that follows. As opposed to the pruning step, columns excluded by preselection are not removed from the dataset; instead, they remain available to subsequent wrangling steps, as they may still become useful features after more transformations.

Feature ranking The aim of this step is to quickly rank columns by potential relevance. To do so, AVATAR uses a wrapper approach—learning shallow models on subsets of features and aggregating the feature importances extracted from these models. In every iteration, n rows are randomly sampled from a random subset of columns. On this subset of data, we perform k-fold cross-validation with a shallow learner $\mathcal{F}_r$. In each of the folds, feature importances are estimated from these learned models, averaged for each column and weighted by the cross-validation performance. Final importances for each column are obtained by averaging the weighted importances over multiple iterations.

To estimate model performance, we use accuracy in case of classification and $\max(0, R^2)$ in case of regression. Feature importances within each model are estimated using SHAP values [12, 11]. They are a practical implementation of the game-theoretic concept of Shapley values, which quantify an individual player's contribution towards the final outcome in a cooperative game [15]. Each feature takes the role of a player and a prediction is considered the outcome of a game.

4.3 Evaluate

Given the ranking of features, AVATAR now heuristically evaluates its progress on the current dataset. It looks for a k such that the top-k ranked features result in the best performance. Performance for a set of features is evaluated using cross validation on all rows of the dataset using a learner $\mathcal{F}_e$. Accuracy is used for classification and RMSE for regression.

If performance decreases with respect to previous iteration, AVATAR terminates and returns the set of features that achieved the highest performance. A user can easily request more features from AVATAR, which are returned in order of their rank. The wrangling program is also generated from these selected features by adding `drop` transformations for columns that are generated but not selected, as was shown in Figure 1b.

4.4 Wrangle

In this step, AVATAR generates new candidate features by transforming the columns of the current dataset. We exhaust the generators for all transformations on all columns that were not wrangled before and apply the transformations to obtain new columns. These new columns are appended to the current dataset, ensuring that more complex features—obtained by applying more transformations—are pruned over simpler ones.

5 Evaluation

We perform experiments to answer the following questions.

Q1 Is AVATAR able to find new and useful features?
Q2 How does AVATAR compare to human wranglers?

Table 2: Data used for evaluating AVATAR.

(a) Classification (C) and regression (R) datasets.

Dataset	Type	Columns		Rows
		Total	Text	
Android	R	10	8	984
Car features	R	17	8	11914
Car price	R	27	10	205
Food choices	R	61	13	121
GSM	R	40	39	8628
House	R	82	43	1460
Melbourne housing	R	22	8	13580
NBA	C	21	7	128069
NBA2K	R	15	11	429
Pet	C	11	4	18834
Shelter animals	C	10	10	26729
Titanic	C	12	5	891
iPhone 11	R	7	3	247

(b) Notebooks compared with AVATAR.

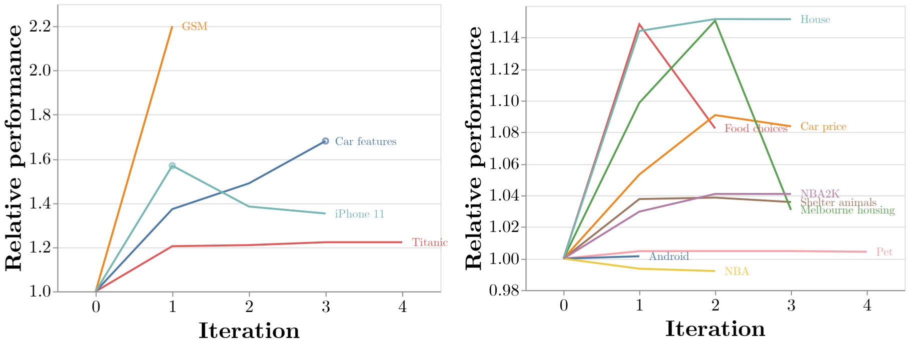
Dataset	Lines of code	# features	
		Human	AVATAR
iPhone 11	28	6	7
NBA	34	10	9
Pet	44	66	10
Car features	42	3	26
Food choices	57	133	17
GSM	277	14	42

Data We use datasets from Kaggle, a popular data science platform. Kaggle allows users to publish datasets and provides *public notebooks* which contain snippets of code executing the data science steps. We search for datasets that (1) contain scraped data, (2) have a single file, and (3) a clear prediction target. We focus on evaluating AVATAR's ability to wrangle interesting features and, therefore, only use the datasets that have at least one column that requires wrangling. An overview of datasets is shown in Table 2a.

Models and metrics As we are interested in AVATAR's ability to wrangle new features, not in obtaining the best possible performance on a dataset, our primary concern when choosing the model for estimating feature importance is its speed (as we need to train it frequently) and ability to identify useful features. We assume that even low-capacity models are capable of identifying useful features, though their estimate might be less robust compared to complex models. For this reason, we focus on decision trees and limit their depth to 4 when evaluating feature importance and to 12 when training the final model. We report the relative performance over the absolute performance.

Experimental setup Experiments were performed on a laptop with a prototype implemented in Python. Code, data and results are available on GitHub.[2] We ran AVATAR for four iterations, with 1600 iterations of feature selection on samples of 1000 rows. In some datasets with many columns containing long strings, the number of columns can quickly explode after a few iterations. If the pruning step took longer than two hours, AVATAR was stopped early.

[2] https://github.com/pidgeyusedgust/avatar-ida21



(a) Performance gain of 20% or more.

(b) Performance gain of less than 20% or performance loss.

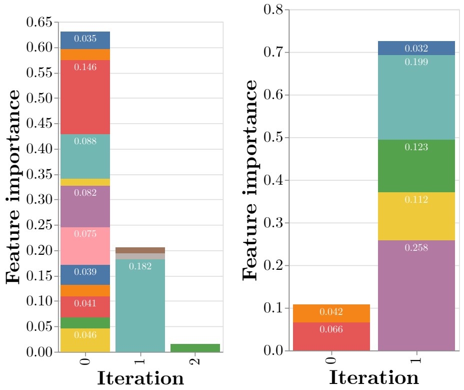
Fig. 4: Relative performance of AVATAR after iterations of wrangling new features when compared to the original dataset (iteration 0). Feature importances for marked data points are shown in Figure 5a.

5.1 Wrangling new features

Evaluating the quality of features is impossible to do directly; instead, we evaluate their quality implicitly through the performance of a model trained on the features. More precisely, we compare performance of the model training the raw data versus the model trained on data wrangled by AVATAR. The relative performance after each iteration is shown in Figure 4. Note that AVATAR starts with pruning and selection, and the baseline result at iteration 0 is thus also obtained after greedily selecting features for the best performance without wrangling.

The results show that AVATAR consistently improves predictive performance by wrangling new features. A single exception is the NBA dataset, where wrangled features are not relevant to the target. This is reinforced in the next experiment, where AVATAR performs on par with human wranglers. We observe a general trend where performance drops after multiple wrangling iterations. The reason for that is the noise in feature importance estimation: our estimate becomes less robust with the increase of the number of features because AVATAR repeatedly uses uniformly sampled subsets of features to estimate their importance. With the increase in the number of features, there is a higher chance of a spurious interaction between the features. This negatively impacts the performance of the final model due to overfitting. AVATAR then terminates and returns the ranked features from the previous iteration.

As a small case study, we take a closer look at the best performing features for the Melbourne housing and iPhone 11 datasets in Figure 5a. One column from their original datasets is shown in Figure 5b. For the Melbourne housing dataset, the dummy encoded feature for "Southern" after splitting **Regionname** on " " is found to be the most relevant one. The selected feature in the second iteration first splits this column by "-" and then extracts the word "Metropolitan",



(a) Feature importances after wrangling for the Melbourne housing (left) and iPhone 11 datasets (right). Each colored segment is one feature.

Description
iPhone 11 256GB - Purple Unlocked
iPhone 11 64GB - Black Unlocked
iPhone 11 Pro Max 512GB - Silver Unlocked
iPhone 11 Pro 512GB - Silver Unlocked
iPhone 11 128GB - Black Unlocked

Regionname
Western Metropolitan
Eastern Metropolitan
South-Eastern Metropolitan
Eastern Victoria
Southern Metropolitan

(b) Original columns from (top) iPhone 11 and (bottom) Melbourne housing datasets.

Fig. 5: A closer look at selected features for two datasets.

"South" or "Victoria". This results in "South-Eastern Metropolitan" and "Southern Metropolitan" being projected to the same "South" feature, which a human might not think of. On the iPhone 11 dataset, many features are extracted from the **Description**. Very relevant is the full model name "iPhone 11 256GB" as obtained by splitting on "-". Human data scientists might expect that this feature requires to be split up further. It is, however, an ordinal feature on its own and provides a strong signal.

5.2 Comparison with humans

In the second experiment, we compare the performance of a predictive model on a dataset wrangled by (1) human experts on Kaggle and (2) by AVATAR. We obtain expert-wrangled datasets from the corresponding notebooks on Kaggle. As we are interested in the ability to wrangle features, any feature engineering steps are removed from the notebooks, but any feature selection is left untouched. A list of notebooks and number of lines of wrangling code is given in Table 2b. We compare the relative performance of the same model trained on features wrangled by humans versus features wrangled by AVATAR and show them in Figure 6.

The results show that AVATAR performs similarly or better than human data wranglers. The only exception is the iPhone 11 dataset. AVATAR still identifies interesting features, but the main reason for bad performance are noisy examples—iPhone 11 covers instead of phones—which negatively impacts the performance of AVATAR. For the NBA and Pet datasets, human performance is marginally better. Wrangled features are not representative of the target, which further explains their small performance differences in Figure 4b. On datasets where AVATAR greatly improves with respect to the baseline, it also beats human wrangling.

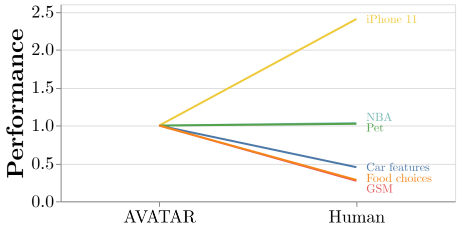


Fig. 6: Comparing the relative performance of human wranglers to AVATAR. Downward slopes indicates that AVATAR is better, which happens for half of the datasets. For the NBA and Pet dataset, the previous experiment has already shown that little additional information is present in wrangled features.

6 Conclusion and future work

In order to cope with data scientists spending valuable time on this tedious process, we present the AVATAR algorithm for automatically wrangling features from raw columns. We show that AVATAR is able to wrangle features that improve predictive performance when compared to the original dataset. On datasets that require heavy wrangling, it even outperforms some human wranglers.

Future work Two immediate pointers for extensions are expanding AVATAR to the multi-relational setting, allowing different tables to be joined, and exploring the unsupervised case, for example, by using multi-directional ensembles of decision trees [16]. Unsupervised data wrangling would allow AVATAR to aid exploratory data analysis, another significant time sink for data scientists. Quickly selecting relevant features from high-dimensional data with high multicollinearity plays an important role in AVATAR. Our repository contains the intermediate, wrangled datasets to encourage further research on this topic. The main technical limitation for AVATAR is that the search space quickly explodes when many columns with long, textual values are present. Being more strict on the generators and pruning rules can trade off speed for expressive power.

Acknowledgements

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No [694980] SYNTH: Synthesising Inductive Data Models). This research received funding from the Flemish Government (AI Research Program). Sebastijan Dumancic is funded by the Research Foundation-Flanders (FWO).

References

1. Bavishi, R., Lemieux, C., Fox, R., Sen, K., Stoica, I.: Autopandas: neural-backed generators for program synthesis. Proceedings of the ACM on Programming Languages **3**(OOPSLA), 1–27 (2019)

2. Dasu, T., Johnson, T.: Exploratory data mining and data cleaning, vol. 479. John Wiley & Sons (2003)
3. Drosos, I., Barik, T., Guo, P.J., DeLine, R., Gulwani, S.: Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. pp. 1–12 (2020)
4. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Auto-sklearn: efficient and robust automated machine learning. In: Automated Machine Learning, pp. 113–134. Springer, Cham (2019)
5. He, Y., Chu, X., Ganjam, K., Zheng, Y., Narasayya, V., Chaudhuri, S.: Transform-data-by-example (tde) an extensible search engine for data transformations. Proceedings of the VLDB Endowment **11**(10), 1165–1177 (2018)
6. Jin, Z., Anderson, M.R., Cafarella, M., Jagadish, H.: Foofah: Transforming data by example. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 683–698 (2017)
7. 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 (2011)
8. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors. In: 2015 IEEE international conference on data science and advanced analytics (DSAA). pp. 1–10. IEEE (2015)
9. Kaul, A., Maheshwary, S., Pudi, V.: Autolearn—automated feature generation and selection. In: 2017 IEEE International Conference on data mining (ICDM). pp. 217–226. IEEE (2017)
10. Le, V., Gulwani, S.: Flashextract: a framework for data extraction by examples. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 542–553 (2014)
11. Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.I.: From local explanations to global understanding with explainable ai for trees. Nature Machine Intelligence **2**(1), 2522–5839 (2020)
12. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in neural information processing systems. pp. 4765–4774 (2017)
13. Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. In: Workshop on automatic machine learning. pp. 66–74. PMLR (2016)
14. Raman, V., Hellerstein, J.M.: Potter's wheel: An interactive data cleaning system. In: VLDB. vol. 1, pp. 381–390 (2001)
15. Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games **2**(28), 307–317 (1953)
16. Van Wolputte, E., Korneva, E., Blockeel, H.: Mercs: multi-directional ensembles of regression and classification trees. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. pp. 4276–4283. AAAI Publications, New Orleans, Louisiana, USA (2018)
17. Yan, C., He, Y.: Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 1539–1554 (2020)
18. Yang, C., Akimoto, Y., Kim, D.W., Udell, M.: Oboe: Collaborative filtering for automl model selection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1173–1183 (2019)