Data wrangling & preparation automation. Why should you lose 80% of your time in one task?

Thesis · June 2021

DOI: 10.13140/RG.2.2.13981.03047

1 author:

Mahmoud Elansary
University of Liège
41 PUBLICATIONS **2,197** CITATIONS

SEE PROFILE

KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS KU LEUVEN

Postgraduate Studies in Big Data & Analytics in Business and Management

Dissertation

Data wrangling & preparation automation

Why should you lose 80% of your time in one task?

Author Mahmoud Elansary

June 2021

Acknowledgements

Throughout the writing of this dissertation, I have received a great deal of support and assistance. I would first like to thank Professor Jochen De Weerdt and Professor Seppe Vanden Broucke for organizing these postgraduate studies sessions in big data, which have increased my knowledge of new aspects of the data science world that I did not know about before. I believe all the sessions were great from organization and timing, and they never forget the networking part with the break-out rooms. It would have been great if I could meet you both in person, but to this moment, many thanks for the dedication and the effort. I felt how much you both are committed to the process mining domain, and I think my next side project should be an application of process mining and I am looking forward to your feedback on it one day.

I want to also acknowledge my colleagues from Cnext for their fantastic feedback and especially Peter Van Kerkhove for allowing me to join this course with financial support from Cnext.

I want to send a special thanks to my beloved wife Noha for taking care of our two kids (Yassin & Selim) while being in the long evening sessions on Fridays. I really appreciate you for providing me with the dinner, including the snacks during the small breaks between the sessions. I would also like to thank Noha again for allocating more time for me to finish typing this manuscript in time, at the expense of her time during this special time in her career.

In addition, I would like to thank my parents as without them. I would not have come to this world. It is hard to describe how they helped and supported me to finish my bachelor studies and to move to Europe for a new chapter in my life.

Finally, I would like to finish my acknowledgment to thank the Elansary and Abouelil family all around the world for being there for me at any moment in my life; even though mountains, seas, and oceans separate us, we are still asking about each other in the family WhatsApp group.

I wish this era of Corona will come to an end this summer as it has taken so many loved ones everywhere around the world to their souls and to my father's soul I dedicate this dissertation.

Table of Contents

Summary

One of the most time-consuming and least rewarding part of a data analytics project is the data preparation phase. As it is a critical piece in any data project and can dramatically affect the project's outcome, most companies will spend the majority of their planned time ~ 80% in this stage. Unfortunately, more projects/tools are being developed to automate the 20 % task – Modeling – by introducing the new concept of AutoML without giving much attention to the dominant time-consuming part in the equation.

This automation is recently shifting in the correct direction towards automation of the data cleansing stage and the data exploration phase. In the past decade, more companies such as HoloClean, Trifacta, Tamr & Inductiv (recently acquired by Apple) started to emerge on the surface by focusing their business model on automating the data preparation task using AI. This was predicted by Gartner back in 2018 as they estimated the market for data preparation tools to be worth 2.9 billion dollars.

ML models are now being used for finding errors in data as well as fixing these errors. Such techniques will fit in the new emerging DataOps agile methodology, which tries to find the best practices and tools to improve the quality and reduce the cycle time of data analytics. Therefore, I would like to explore the different possible automation tools/packages in the last five years for data preprocessing to reduce the wasted time in the manual cleaning of data. I will also try to explore some of the data drifting techniques, as being able to detect early drift in the data will lead to more trusted models, which will help achieve better decision-making.

Abstract

A data science project is a long journey that starts by collecting and gathering needed data for a particular use case. Usually, these use cases have a dominant business component that is driving the whole project.

Based on my experience in a Belgian consulting firm that had a mission for helping clients to use their collected data towards data science, use cases to improve relevant business KPIs. I would say that usually, we get access to client data very early during our engagement for a demo as clients typically want to see what you can do with their data from the second meeting. This would not be a trivial task even if you had a use case in mind from the first meeting, as the data cleaning and preparation part can get quite lengthy, especially in small data science teams. That is why we need to have tools ready to jump-start the demo phase to get the data prepared for a quick analysis as soon as possible.

In this dissertation, I try to present my thoughts like a review paper for the latest state-of-the-art python packages appearing in the last five years. I try to focus more on packages dedicated to data preparation and data exploratory automation. Usually, having these automation tools/packages in your arsenal helps in speeding and easing your task. I choice packages written in python as this is the most dominant language used outside the academic world. The list of packages reviewed in this paper is not exhaustive, but it will cover most of the essential steps in the data preparation phase.

1 Introduction

1.1 What is data preparation?

Data preparation is the method of cleaning, reformatting, and transforming raw data earlier to be ready for the analysis phase. Data preparation is a critical step in a data science project life cycle. It is a fundamental step which will affect the results and performance of the whole project, that's why data scientist will tend to invest a significant portion of their time making sure the data is ready for the analysis phase. Data preparation is regularly a lengthy undertaking for data professionals, experts, or business users. Nevertheless, it is a step that cannot be skipped or passed quickly without much attention. Having clean data is an essential precursor to reaching an accurate and efficient business decision. According to Talend, a famous software publisher [1], data preparation help to achieve three main goals:

1. They are fixing errors rapidly, data preparation aids in the detection of errors before processing. These inaccuracies become more challenging to understand and remedy once data has been removed from its source.

2. They produce top-quality data, as datasets are cleaned and reformatted to ensure that all data utilized in the analysis is of good quality.

3. Making better business decisions, as more timely, efficient, and high-quality business decisions result from higher-quality data can be handled and evaluated more quickly and efficiently.

Furthermore, companies started to move towards cloud computing as it had got cheaper during the past decade, and it also has other advantages that come with it. One of the benefits of switching to the cloud is scalability, as now the infrastructure will never limit the amount of data a company needs to store. The emerging data lake concept made it easy to store unstructured and structured files in the same location. This is mainly provided by the three leading top vendors in this domain (Microsoft, Amazon, and Google). Having structured dataset files in the data lake makes the starting of the data preparation process easier as most of the raw data is ready in the cloud. The next step is to connect your data lake to an online compute cluster such as Azure machine learning cluster or Databricks cluster. Databricks support spark clusters, making the data preparation phase much faster when you have substantial big datasets. Databricks has introduced the concept of delta tables which is an alternative to the old lambda architecture. The Delta architecture adds a new abstraction layer on the data lake that gives the user a long list of advantages. I think data versioning and schema enforcing are powerful concepts that can help data users a lot. From Figure 1, we can see the detailed list of advantages of delta architecture.

Figure 1 – *Databricks Delta Lake benefits, source* [2].

The delta architecture consists mainly of three types of tables bronze, silver, and gold. The concept is straightforward, all your raw data goes to bronze tables, and once you clean your tables with the necessary data processing steps. Then we need to save the new intermediate data to the silver tables. Finally, silver tables create new gold tables that are tables in a tidy verse format ready for analysis or BI reporting. An overall representation of the delta architecture is shown in the following figure.

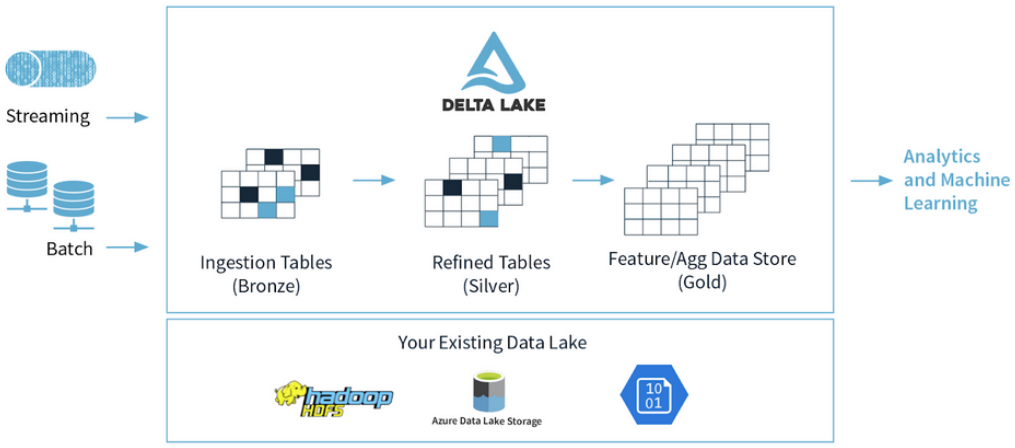


Figure 2 – *The Delta Architecture data quality flow, source* [3].

1.2 Gather data

It all starts with gather data for a data science project or being a data-centric company or getting ready to explore the potential of machine learning new algorithms which can increase the performance of the financial KPIs in a company. Based on my limited knowledge in the consultancy domain, I could say that collecting the data from the client or having access to the correct data is one of the most challenging tasks to prepare a proof-of-concept exercise at the beginning of the engagement with the customer. It is all utterly dependent on the level of maturity the customer data has reached. Some customers want to try using data science techniques without having data, making it impossible to start. In this case, we must suggest to them to wait and start collecting relevant data puting in mind the use case because, in many cases, the client is even lacking a use case for the data he is collecting. This phase of the data science project is an essential prerequisite for the project to start. This is shown in Figure 3 with the CRISP-DM methodology. This problem had been highlighted by a Gartner's report, where they mention that 87% of organizations worldwide have a low BI and analytics maturity level. Usually, this is due to a low data maturity level as well [4]. For this dissertation, I will not go into details about collecting data for a data science project. Still, I will use the available datasets from the famous data science websites to highlight some of the functionality of the latest python packages.

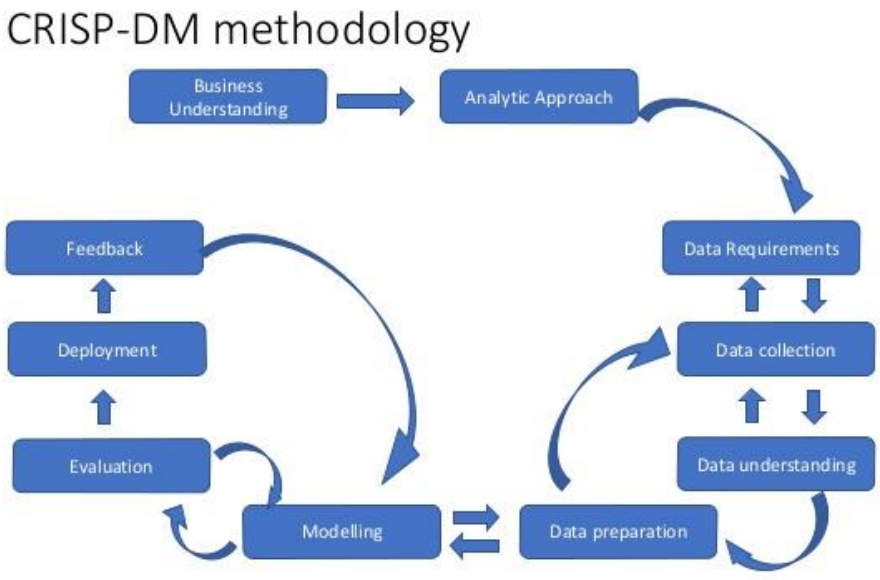


***Figure 3** – CRISP-DM methodology, source* [5].

1.3 Exploratory Data Analysis

Usually, a data scientist or a data engineer does not start cleaning and preparing his dataset write away from the first moment. There is a step or a phase that precedes data preparation, and this is exploratory data analysis. This phase is also as important as the execution of the data preparation steps. During this exploratory phase, the data scientist or the data engineer gets more familiar with the data. We would often use statistical approaches and data visualization to summarize the main characteristics of the different features. The main steps involved in the data exploratory stage can be checked in Figure 4. After this stage, the data scientist or data engineer will feel which data cleaning/preparation steps would be needed for the dataset in hand. Sometimes we will also choose which type of algorithm families would be required for the model. Vendors are starting to give more attention to this step by creating special tools to help speeding up this process, some famous tools that can be used are Trifacta, Qlikview, Weka, KNIME, and the list goes on. These tools can also get your data ready for the modeling phase, as a matter of fact, this vital part of the data science project is starting to have more and more tools in this spectrum. According to Gartner prediction in 2018, the market for data preparation tools is worth 2.9 billion dollars, and it is growing [6]. In this dissertation, I will go through some of the tools which can make the data exploration faster, but I will cover only open-source python packages and not the commercial tools.

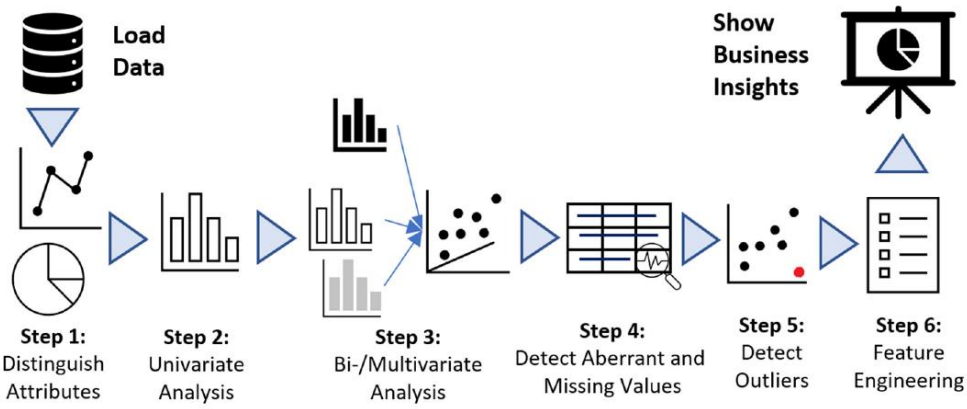


Figure 4 – *Exploratory Data Analysis (EDA) steps, source* [7].

1.4 Steps of Data preparation

The data preparation steps will be different mainly due to the difference in the input data structure. Structured datasets will undergo specific preparation steps to be cleaned compared to unstructured data such as images, text, etc. According to which type of unstructured dataset you have in your project; the preparation steps will change. For this dissertation, I would only cover structure data cleaning steps. Structure data cleaning and preparation can be divided into multiple stages. Some steps can be skipped depending on your dataset structure or the final model that would be used. Let us quickly go over these steps according to multiple blogs and papers; those steps can be summarized as follows, handling missing data, removing outliers, or reducing their effect, removing duplicate records, handling categorical features, and numeric features that need transformation and scaling. Finally, the last step would be generating new features as part of the feature engineering phase. I will not go extensively over the content of these steps in this section as this will be part of the results section of this dissertation.

1.5 Data preparation pipeline

As I mentioned in the previous section, the data cleaning/preparation consists of multiple steps, and once all the steps are ready, we will need to execute them. The best thing is to stitch all the steps into one data preparation pipeline. This will help later whenever we need to retrain the model and, once the model is ready for production, we will need to score new records. Those records will need to follow the same data cleaning steps to be scored by the current model in the production environment. Multiple tools are available in the market at this moment to help with this task. Azure data factory, Azure machine learning services, and Apache airflow are famous tools that allow the creation of data preparation pipelines. Usually, the data preparation pipeline tools are based on a GUI canvas, making it easier to create complicated pipelines with different steps. Figure 5 shows an example of how this would look like in an azure data factory, where a list of notebooks running on azure Databricks are being called inside an azure data factory pipeline. A comparison between these tools and an overview of them would be good future work to consider. The final goal of all these new tools is to make it easier and faster for the data scientist to execute the data preparation steps and monitor them. In azure data factory, when a pipeline is triggered, you would be able to check the progress of each step. Suppose an error appears in the middle of the pipeline. In that case, the data scientist or the engineer can fix the part with the error and then resuming the pipeline again without starting from the beginning again.

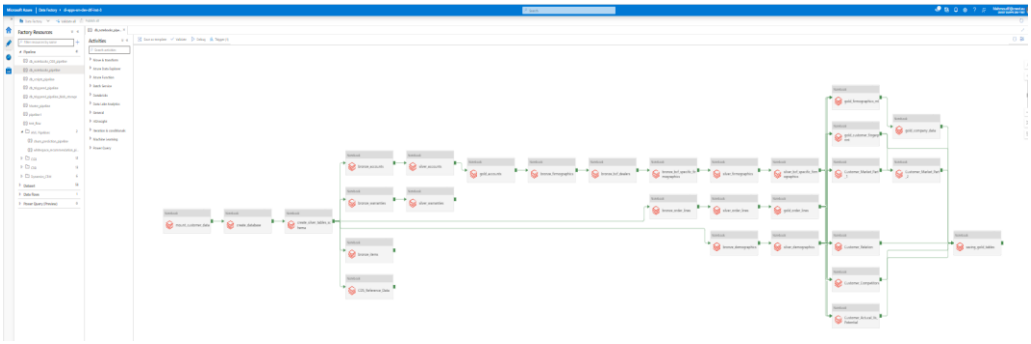


Figure 5 – *Azure data factory data preparation pipeline example.*

1.6 Data drifting

Finally, the model is in production, and the data science project life cycle reached its need. Unfournetly this is often not the end of the story, since after some months or less, depending on the information stream and the change in the domain. The model can start to drop in accuracy either due to concept drifting or data drifting, and both will lead to model drifting. If the target variable changes due to a shift in behavior like for example, fraudulent attacks changing as the hackers changed their techniques, this is called concept drift, and it is a type of model drifting, while if the dependent variables change in the distribution or properties, this will lead to data drifting. Early detection of such drifts is not always easy and can require sophisticated supervised machine learning models to detect drift. As models in production will perform the prediction but will be missing the ground truth so if the model is predicting if a person who gets a loan will default or not, the bank will not be able to find the true value and compare it to the prediction, this will only be possible at the end of lifetime of the loan. So, to be able to know if a model is drifting or not, the MLOps team will tend to monitor the changes in target variable distribution and the changes in the production features. We can use a statistical method to compare the difference between two distributions (baseline and predict) such as Kullback–Leibler, Jensen-Shannon, or Kolmogorov-Smirnov test. In other cases, we will prefer to use a model-based approach to find the drift. To better understand the full picture, review the below figure for MLOps best practices by Microsoft. I will try to review some of the possible python packages that can be used for this task in the last part of the results section.

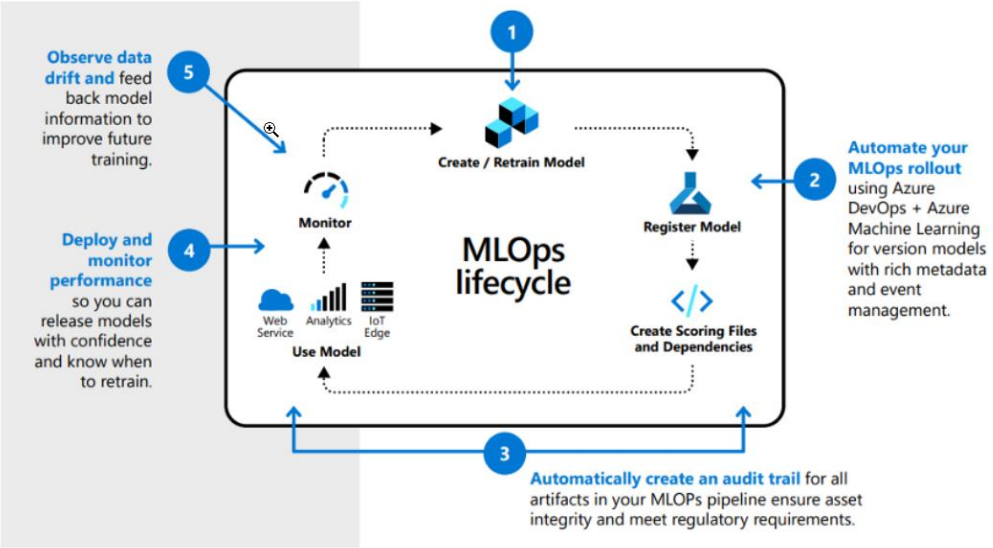


***Figure 6** – MLOps infographic by Microsoft* [8].

1.7 Data cleaning steps automation

At the beginning of my introduction, I have mentioned that the data preparation part is the most time-consuming in the data science project life cycle. This was partially due to the lack of advanced tools for data exploration and data preparation. Luckily, this is not the case in the last decade as more tools from different vendors are avaliable. These tools are usually not free, and because of this, they are still not being used fully by all data scientists, especially in small startup companies. Traditionally data scientists in small teams would prefer to use open-source libraries written in R or python. Unfournetly the advancement in these packages and tools was not as fast as the premium software dedicated to this task. Since the open-source community became more advanced and started new projects towards automating the different steps in the data preparation process and automation of the entire end-to-end life cycle of the whole data science project. I want to shed some light on the latest python packages that have been developed in the last five years. This would help data scientists be aware of the latest community projects, cutting the enormous time spent manually programming some of these steps.

2 The state of the art and Results

2.1 Introduction

This section is dedicated to showing the latest python packages developed in the last five years to automate the data preparation and data exploratory analysis. The primary purpose will be to display the main functionalities of these tools/packages and, when possible, to use available online datasets for this task. This section will be divided into eight sections, data exploratory analysis top 4 tools, then six sections for the data preparation steps: data imputation, outlier detection, deduplication, categorical and numerical transformation and finally the last part of the data preparation would be consisting of two subsections for the two top feature engineering packages. The last part for the closing will touch upon the concept of data drifting, which is covered marginally for completeness.

2.2 Data Wrangling/Exploratory Data Analysis

Whenever a data scientist starts working on a new project, he will need to get more familiar with the dataset. This is usually done by exploring and checking the quality of the dataset at hand. At this stage, the scientist would like to explore all the features from summary statistics to visualization. Such a task is usually a manual effort that needs to be repeated every time a new dataset is introduced to the data team.

Exploratory Data Analysis (EDA) is an essential step before executing any deep learning or machine learning algorithm. EDA makes use of a large number of techniques which is being used to perceive various aspects of the data. Most of these techniques will end up as being a step in the data preparation pipeline.

EDA would help the data scientist dissect, sum up the fundamental qualities and explore a dataset via information representation approaches. During this phase, we try to empower all information on all the factors in a dataset and their connections. The ultimate goal of an exploratory data analysis is to uncover significant elements, information representation approaches, discover designs inside information, get links, and find new bits of knowledge [9].

More and more projects from the python community are emerging to speed up this manual process which can save hours and hours of exploration to a few minutes. I will go quickly over some of the best python tools, which makes data exploration easier.

2.2.1 D-tale

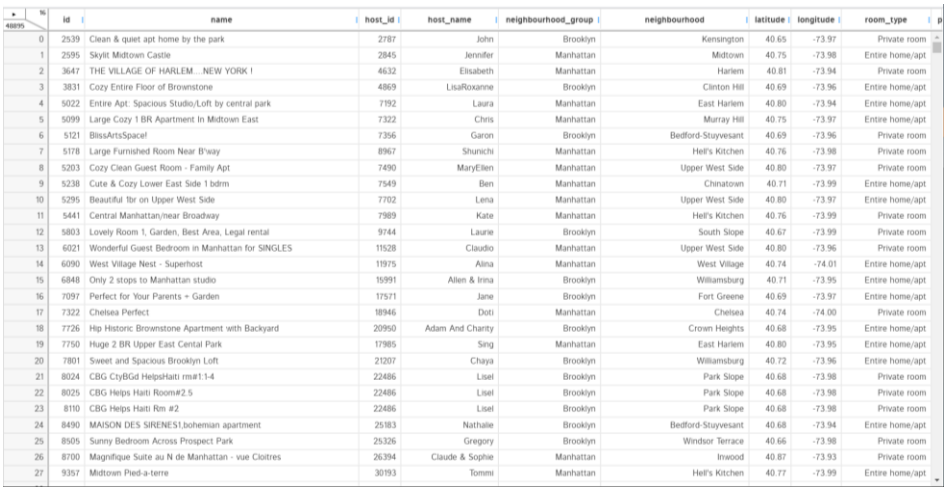
In September 2019, Dtale library [10] was launched to help data scientists in data wrangling and visualization of datasets. It has quickly been downloaded over 500K times in this short period of time. The library is using reacts as a frontend and flask as a backend. It has a vast number of features to reduce the time spent in data exploration. It supports heat maps, 3d plots, interactive plots, builds custom columns, the correlation between features, and many more. It has become so popular quickly that it has already been downloaded half a million times in less than one year.

It is straightforward to start working with, and you need two lines of code (where the df is your pandas DataFrame containing your dataset).

```
d = dtale.show(df)

d.open_browser()
```

It will show in your default browser, as shown in the below figure.



	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.65	-73.97	Private room
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75	-73.98	Entire home/apt
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.81	-73.94	Private room
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.69	-73.96	Entire home/apt
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.80	-73.94	Entire home/apt
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.75	-73.97	Entire home/apt
6	5121	BlissArtsSpace!	7356	Garon	Brooklyn	Bedford-Stuyvesant	40.69	-73.96	Private room
7	5178	Large Furnished Room Near B'way	8967	Shunichi	Manhattan	Hell's Kitchen	40.76	-73.98	Private room
8	5203	Cozy Clean Guest Room - Family Apt	7490	MaryEllen	Manhattan	Upper West Side	40.80	-73.97	Private room
9	5238	Cute & Cozy Lower East Side 1 bdrm	7549	Ben	Manhattan	Chinatown	40.71	-73.99	Entire home/apt
10	5295	Beautiful 1br on Upper West Side	7702	Lena	Manhattan	Upper West Side	40.80	-73.97	Entire home/apt
11	5441	Central Manhattan/near Broadway	7989	Kate	Manhattan	Hell's Kitchen	40.76	-73.99	Private room
12	5803	Lovely Room 1, Garden, Best Area, Legal rental	9744	Laurie	Brooklyn	South Slope	40.67	-73.99	Private room
13	6021	Wonderful Guest Bedroom in Manhattan for SINGLES	11528	Claudio	Manhattan	Upper West Side	40.80	-73.96	Private room
14	6090	West Village Nest - Superhost	11975	Alina	Manhattan	West Village	40.74	-74.01	Entire home/apt
15	6848	Only 2 stops to Manhattan studio	15991	Allen & Irina	Brooklyn	Williamsburg	40.71	-73.95	Entire home/apt
16	7097	Perfect for Your Parents + Garden	17571	Jane	Brooklyn	Fort Greene	40.69	-73.97	Entire home/apt
17	7322	Chelsea Perfect	18946	Doti	Manhattan	Chelsea	40.74	-74.00	Private room
18	7726	Hip Historic Brownstone Apartment with Backyard	20950	Adam And Charity	Brooklyn	Crown Heights	40.68	-73.95	Entire home/apt
19	7750	Huge 2 BR Upper East Cental Park	17985	Sing	Manhattan	East Harlem	40.80	-73.95	Entire home/apt
20	7801	Sweet and Spacious Brooklyn Loft	21207	Chaya	Brooklyn	Williamsburg	40.72	-73.96	Entire home/apt
21	8024	CBG CtyBGd HelpsHaiti rm#1:1-4	22486	Lisel	Brooklyn	Park Slope	40.68	-73.98	Private room
22	8025	CBG Helps Haiti Room#2.5	22486	Lisel	Brooklyn	Park Slope	40.68	-73.98	Private room
23	8110	CBG Helps Haiti Rm #2	22486	Lisel	Brooklyn	Park Slope	40.68	-73.98	Private room
24	8490	MAISON DES SIRENES1,bohemian apartment	25183	Nathalie	Brooklyn	Bedford-Stuyvesant	40.68	-73.94	Entire home/apt
25	8505	Sunny Bedroom Across Prospect Park	25326	Gregory	Brooklyn	Windsor Terrace	40.66	-73.98	Private room
26	8700	Magnifique Suite au N de Manhattan - vue Cloitres	26394	Claude & Sophie	Manhattan	Inwood	40.87	-73.93	Private room
27	9357	Midtown Pied-a-terre	30193	Tommi	Manhattan	Hell's Kitchen	40.77	-73.99	Entire home/apt

Figure 7 *– Dataset shown in the browser after dtale call.*

Once you hover your mouse over any column heading, a drop-down menu will show up, and you would be able to choose from extensive options like sorting the data, describe the dataset, column analysis, and many more.

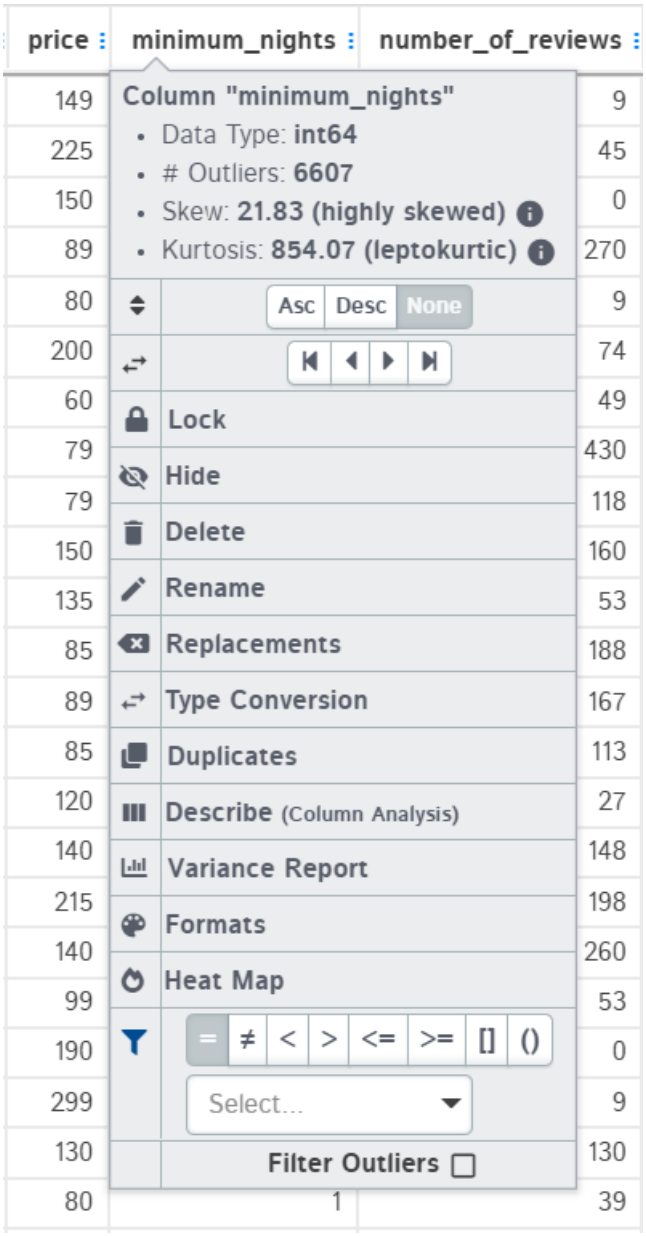


Figure 8 – *Dtale per column option set.*

Once pressing on the “Describe” option from the previous window, then we will get a list of possible summary statistics from the selected feature like min, max, mean, median, standard deviation, variance, quantiles, and many more.

You can also select between different types of graphs, and you have the option to export the code at any time in case you want to modify the plot or customize it more according to your dataset. This makes it much faster to modify the existing plot code rather than starting from scratch.

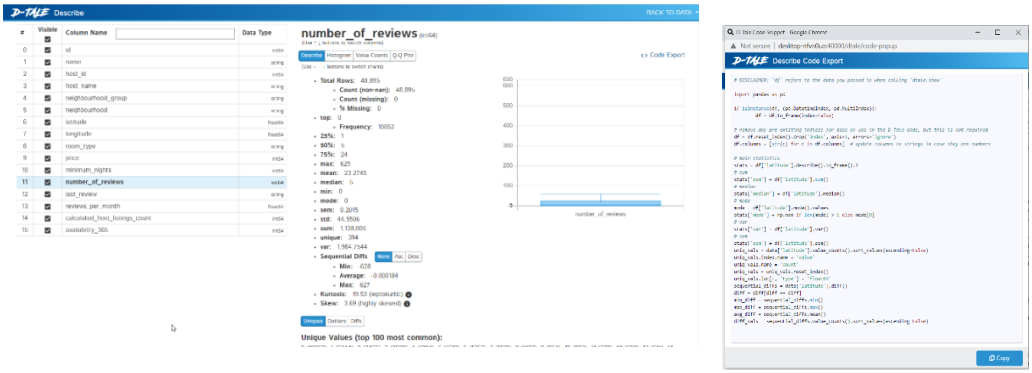


Figure 9 – *selection between different graphs for a given feature.*

I will not go over all the various elements of this library. My intention is not to make an extensive summary of this package. Still, the user will have a strong arsenal of all possible data summary and exploration techniques with a few number of click.

A Quick features summary:

- Showing and removing duplicates

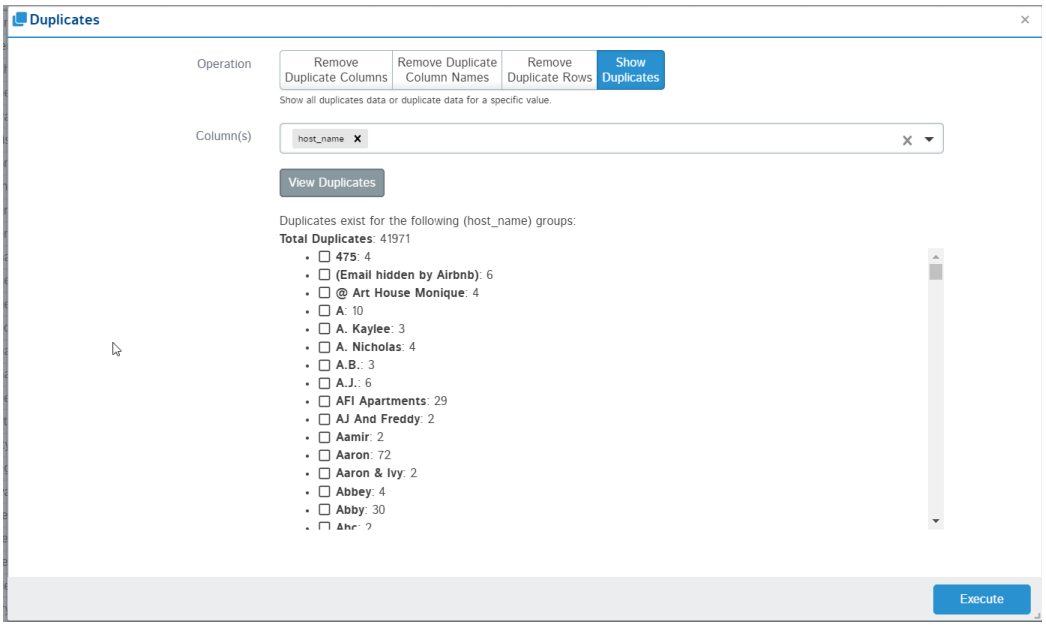


Figure 10 – *duplicate removing window.*

- Correlations

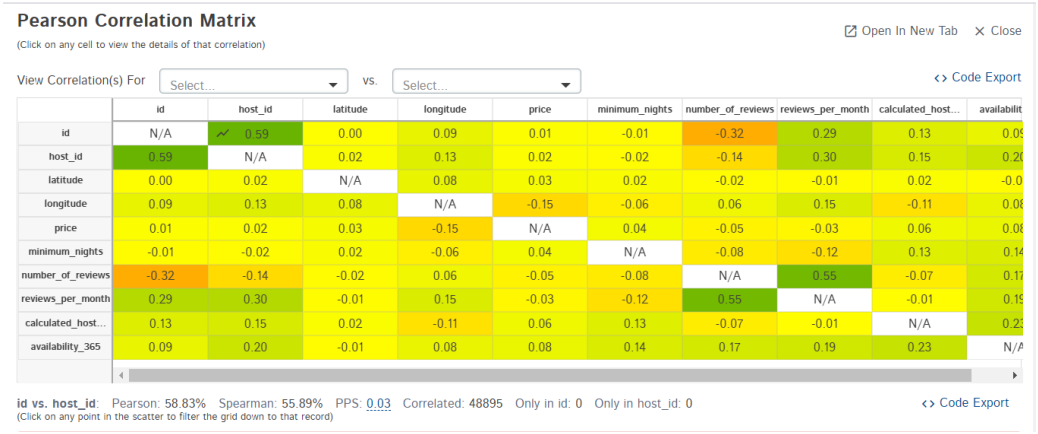


Figure 11 – *Correlation matrix window.*

- Charts

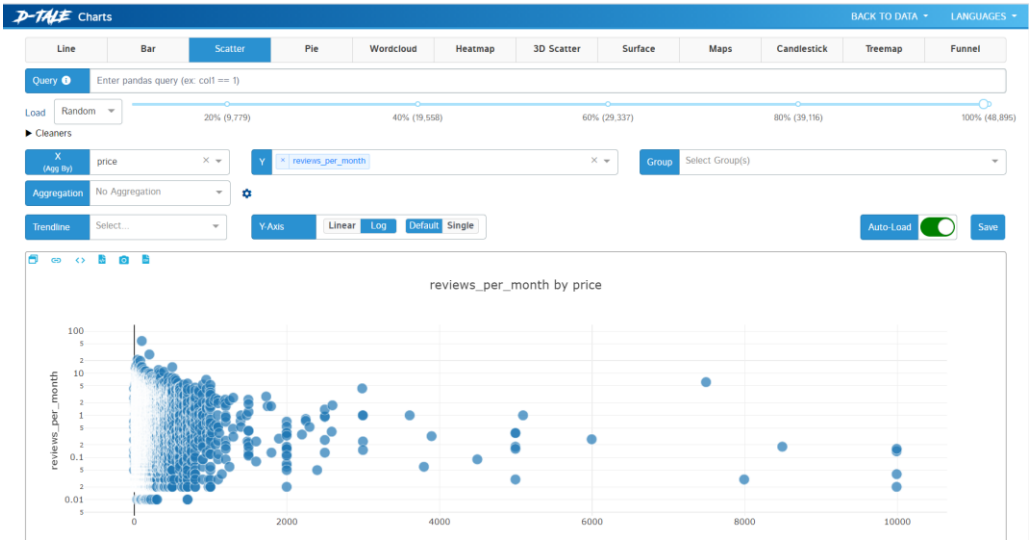


Figure 12 – *charts panel.*

- Grouping, pivot, and transpose

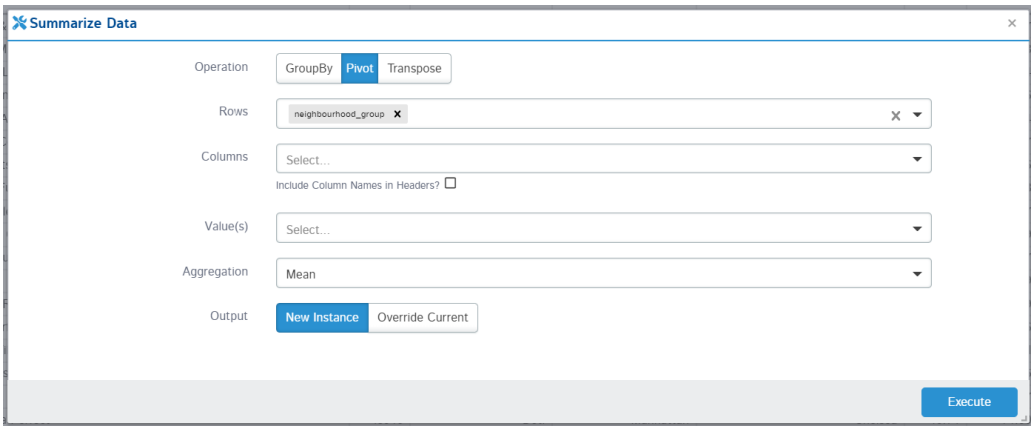


Figure 13 – *summarize data window.*

The user can see the extended list of potential library features by pressing on the top left corner of the tool in the browser.

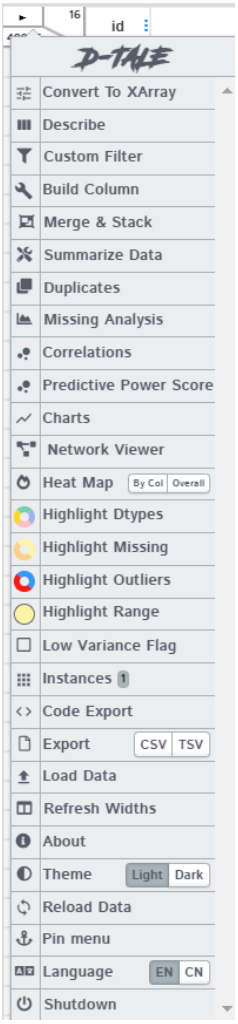


Figure 14 – *Dtale – list of all possible features.*

2.2.2 Pandas-Profiling

This library is an open-source package for python, and it produces interactive HTML reports to summarize the various aspects of the dataset. The pandas-profiling package was first released in May 2016, and it has now passed the five million downloads as it has been the only sort of automated EDA tool for python for some years. The main strength of this package is that it can handle missing values, show summary statistics like mode, mean, median, variance, skewness, kurtosis, etc. Basic charts are also offered, such as correlations plots and histograms.

In the following figures, we can see a glimpse of what the pandas-profiling HTML report looks like using a sample dataset. The report displays how many variables are in our dataset, missing cells per column, percentage of missing cells, the total number of rows, number, and percentage of duplicate rows. Another essential thing that the report will show is the full size of the memory of the dataset. On the right-hand side of the output screen, we will see the different variable types.

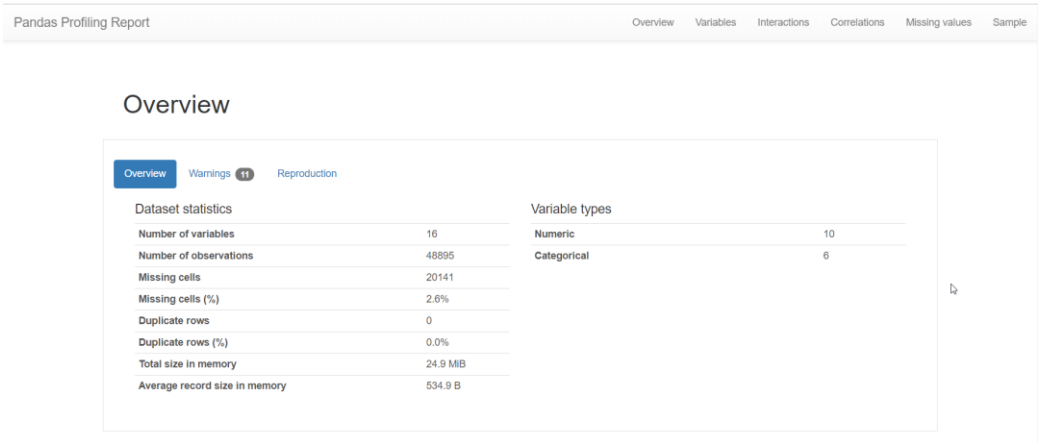


***Figure 15** – Pandas Profiling Report.*

The analysis of a particular feature will appear in the variable section. For example, for the categorical variable, the subsequent output will be displayed.

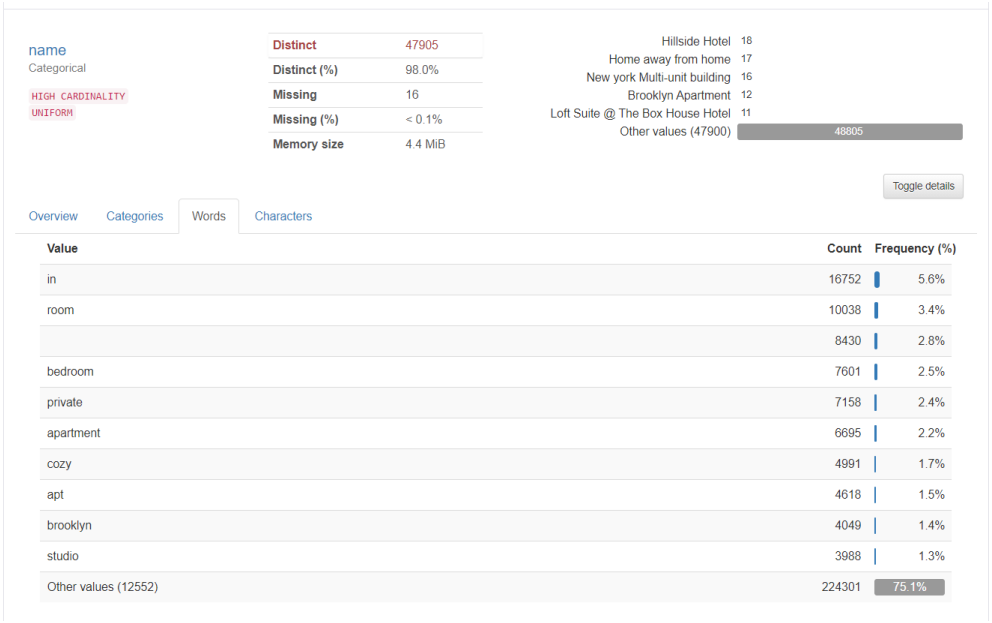


Figure 16 – *summary statistics for a categorical variable.*

While for a numeric column, the below output will appear.

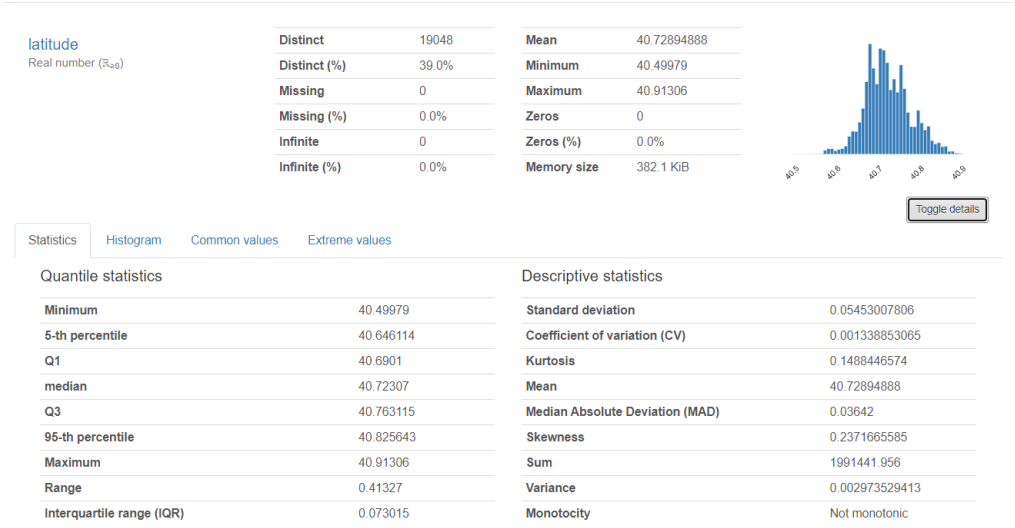


Figure 17 – *summary statistics for a numeric variable.*

A quick summary of the available features:

- Correlation

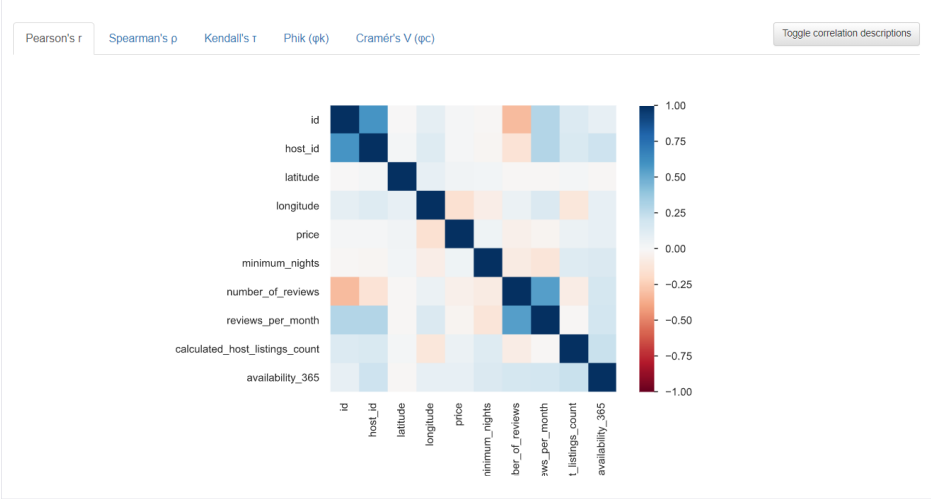


Figure 18 – *pandas profiling correlation matrix.*

- Missing values

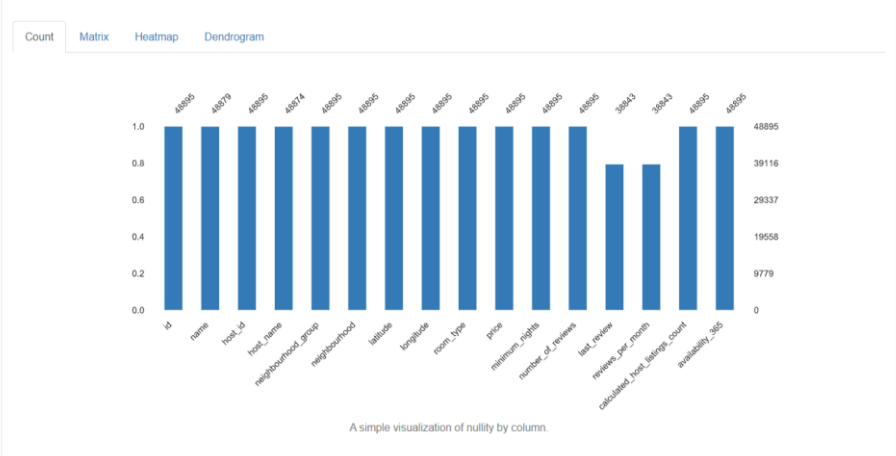


Figure 19 – *pandas profiling missing value report.*

- Overview

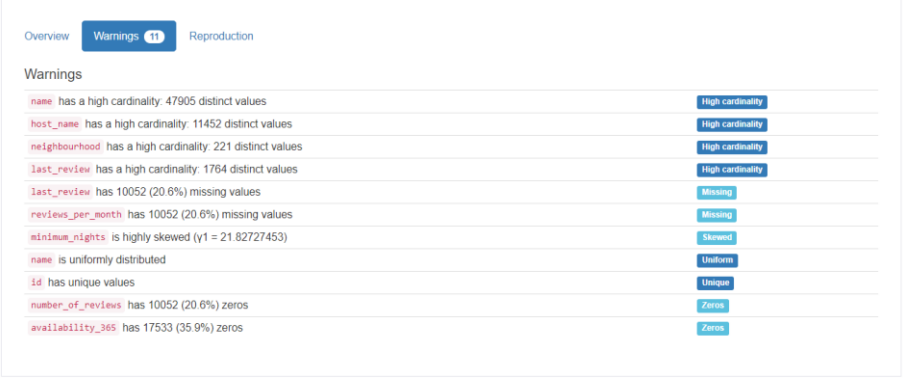


***Figure 20** – pandas profiling warnings overview.*

2.2.3 SweetViz

Another new python package dedicated to jump-start your EDA is the SweetViz package [11]. This one is instead a recent one from June 2020 and with a total of 150 K downloads so far. It contains most of the features appearing in the last two discussed packages. The project was in significant part inspired by the pandas-profiling package.

***Figure 21** – SweetVIZ main report.*

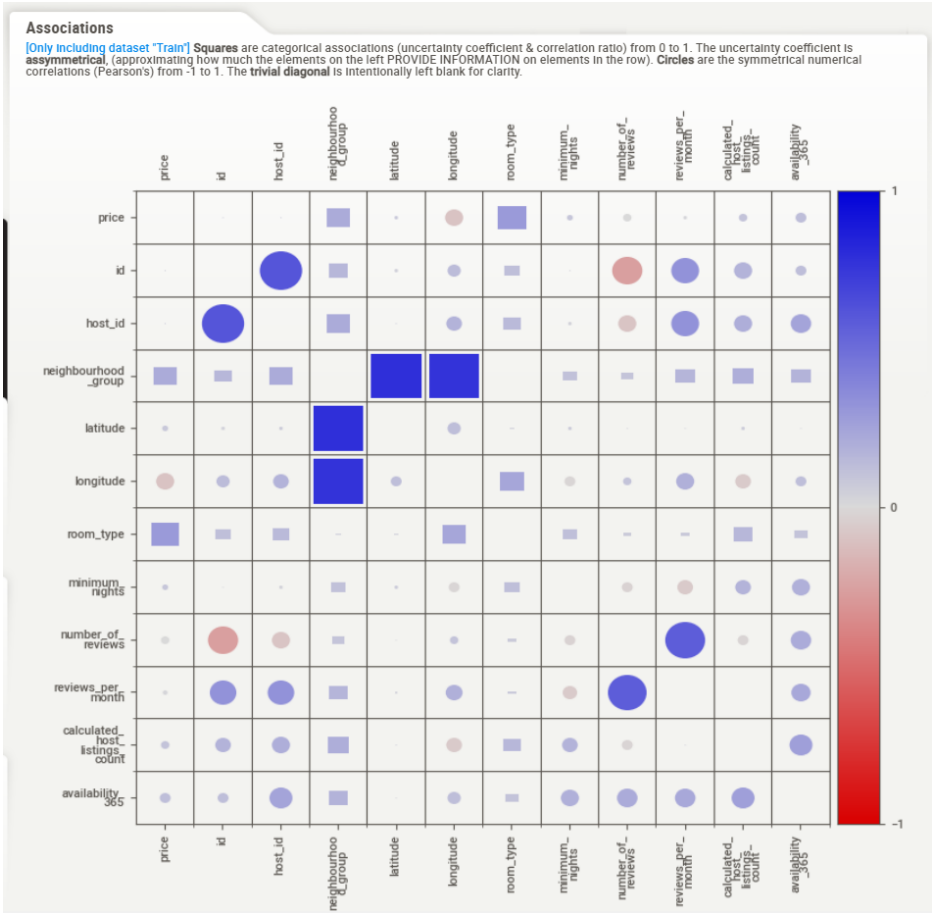


Figure 22 – *SweetVIZ feature association plot.*

2.2.4 AutoViz

The final library under this category is mainly for data visualization. The autoviz package was first released in the summer of 2019 and it has been downloaded over 70k times so far. For more information, check official documentation [12]. Below some figures which shows how the plots will look like when using autoviz on your dataset.

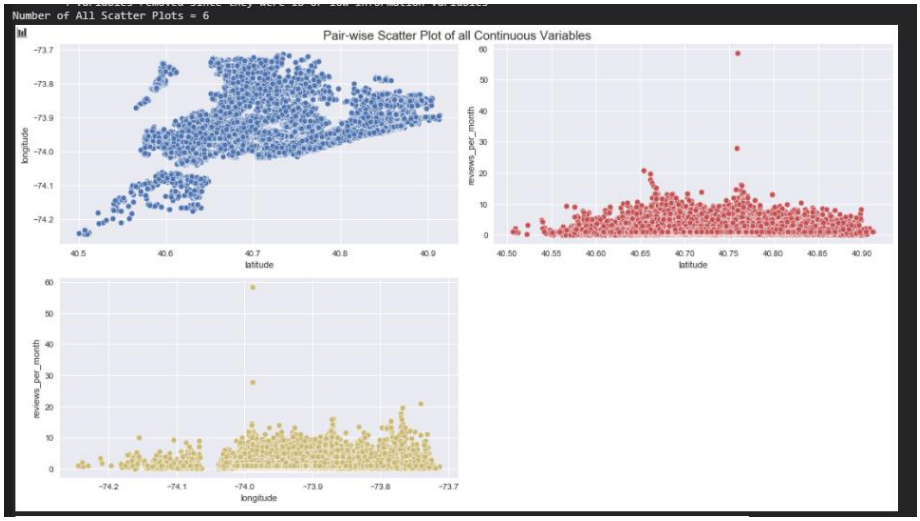


Figure 23 – *AutoViz Pairwise scatter plot.*

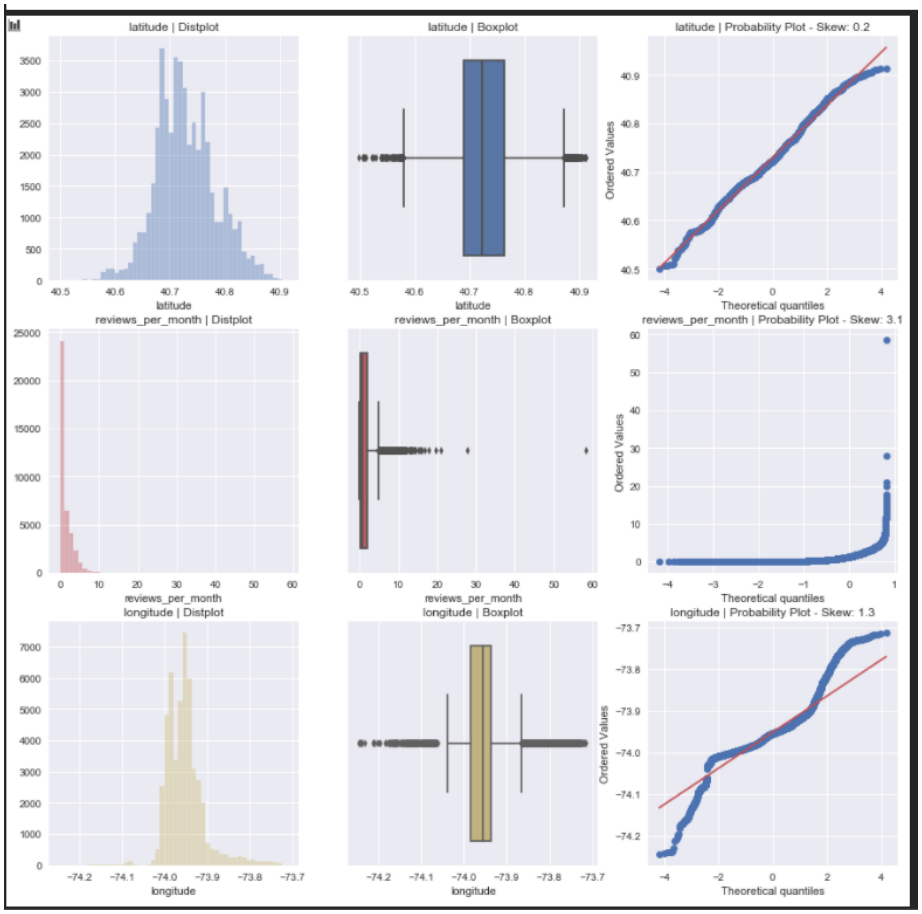


Figure 24 – *AutoViz Pairwise distribution plots.*

2.3 Handling missing data

The Real-world data are often messy and dirty, to handle missing data, we can either perform deletion or imputation. Depending on the size of the dataset, we can either consider deletion if the missing percentage is tiny compared to the size of the data set. Still, we cannot just drop all these records in any use case, or else we will end up with too little data to train our model. In such cases, imputation or filling in the missing values is a must. A quick summary of the process of handling missing data can be seen in Figure 25. Traditionally, straightforward imputation methods like simply using the column average or median do not work well.

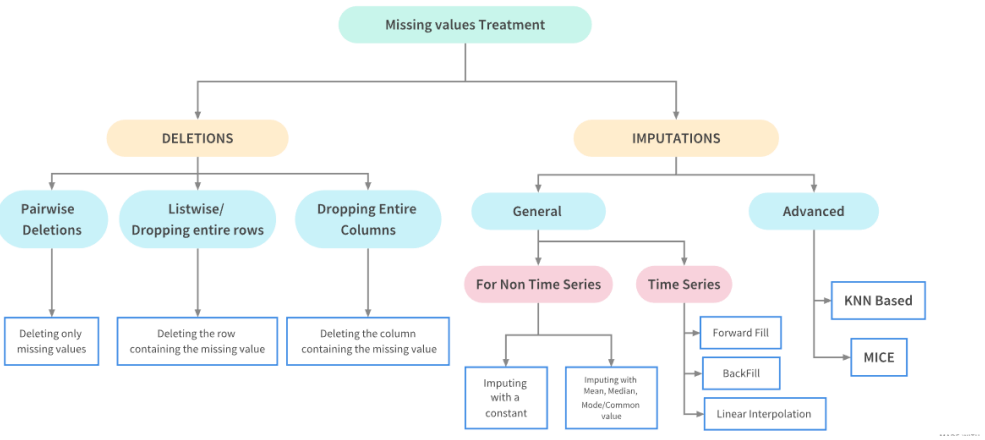


***Figure 25** – handing missing values, source* [13].

On the other hand, using imputation algorithms based on machine learning algorithms such as KNN can be an excellent alternative to lazy methods. Still, we will require the tuning of k parameter. KNN has many weaknesses and vulnerabilities, like being very sensitive to outliers and noise in the data. It will also be computationally expensive depending on the size of the dataset in hand because it would require the entire dataset to be stored and computing distances between every pair of points.

To overcome the disadvantages mentioned above of a KNN imputation approach, another possible algorithm based on the random forest is the MissForest. In 2011 the creators of the MissForest algorithm conducted a study in which they compare different behavior of imputation algorithm on a group of datasets with randomly introduced missing values. They concluded that MissForest was a supreme method with an overall marginal improvement reaching 50% over the KNN-Impute [14].

The MissForest can be easily called on your dataset using the missingpy package from python; the algorithm loops through the missing data points several times. It continues for a certain number of iterations or stops once predefined stopping criteria are reached. For most datasets, a rule of thumb is that we would need between four and five iterations until the algorithm converges, but this would highly depend on the amount and size of

the missing data. The immense added value in using MissForest is that users can apply it to mixed data types such as categorical and numerical data simultaneously [15].

To quickly check this algorithm performance, I have used the famous Iris data set where I randomly chose some rows inside the dataset to set as missing. In this case, I can compare the results of the imputation algorithm to the original missing locations. A comparison between MisForest, K-NN and simple mean imputation is shown in the below figure. MissForest outperforms the trivial imputation task and was slightly better than K-NN, keeping in consideration that K-NN needed more time for running. It is effortless to execute in python with just three lines of code, and it does not need tunning.

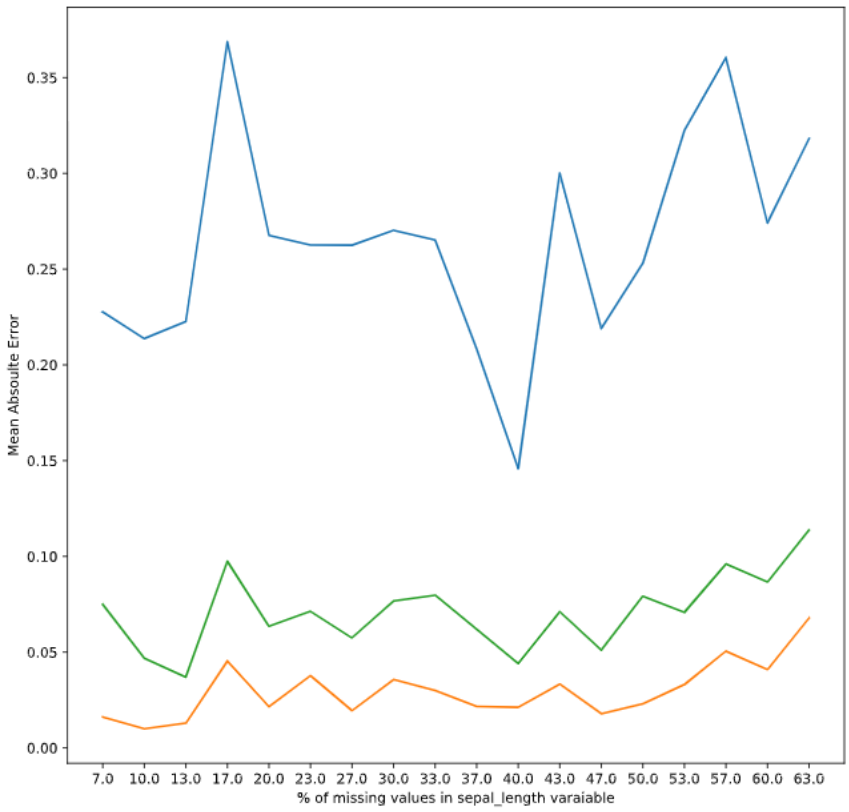


Figure 26 – *Compare Mean Absolute Error for three imputation methods (Mean imputation in blue ,KNN (k=5) in green , and MissForest in orange).*

Now let us quickly mention a relatively new package called Autoimpute, which has implemented multiple different algorithms for imputation. The package was first released in spring 2019, and since then, it has 50k downloads. Autoimpute is an advanced library that has various techniques for handling missing data. The techniques used by Autoimpute is divided into four types:

1. Univariate imputation: We impute the values of the missing positions only based on the current column without using any information from other columns.

2. Multivariate imputation: In this case, we would use additional variables to impute the missing values. We can do this in a linear regression approach.

3. Single imputation: As the name implies, we would only impute the missing value in each location of the variable within the dataset only once. Therefore, we would create a single new imputed dataset from the original dataset.

4. Multiple imputation: This final approach would call the single imputation on the missing dataset multiple times, producing various possible values for a given missing value.

An example of the imputation methods supported by Autoimpute can be found in the table below.

Table 1 – *Autoimpute supported imputation algorithms, source* [16].

Univariate	Multivariate	Time Series / Interpolation
Mean	Linear Regression	Linear
Median	Binomial Logistic Regression	Quadratic
Mode	Multinomial Logistic Regression	Cubic
Random	Stochastic Regression	Polynomial
Norm	Bayesian Linear Regression	Spline
Categorical	Bayesian Binary Logistic Regression	Time-weighted
	Predictive Mean Matching	Next Obs Carried Backward
	Local Residual Draws	Last Obs Carried Forward

Autoimpute fits directly into scikit-learn machine learning projects. Imputers inherit from sklearn's BaseEstimator and TransformerMixin and
implement fit and transform methods, making them valid Transformers in any sklearn pipeline.

```
# create a complex instance of the SingleImputer      You, seconds ago • Uncommitted changes
# Here, we specify strategies by column and predictors for each column
# We also specify what additional arguments any `pmm` strategies should take
imp = SingleImputer(
    n=10,
    strategy={"X1": "pmm", "X2": "bayesian binary logistic", "X3": "norm"},
    predictors={"X1": "all", "X2": ["X1", "X4", "X3"]},
    imp_kwgs={"pmm": {"fill_value": "random"}},
    visit="left-to-right",
    return_list=True
)

# Because we set return_list=True, imputations are done all at once, not evaluated lazily.
# This will return M*N, where M is the number of imputations and N is the size of original dataframe.
imp.fit_transform(data)
```

Figure 27 – *Autoimpute code example.*

2.4 Outlier detection

The presence of outliers in any dataset can significantly affect the performance of many machine learning algorithms applied to this dataset. To eliminate this effect, we would need to either drop outliers from the dataset or bound the value of the outliers to some reasonable value. This value is based on domain knowledge. Finally, the other option in handling the outliers present in the dataset is by transforming the dataset. Usually, our problem is not what to do with outliers once detected but rather how we can detect them from the beginning. Detecting outliers is not always a trivial task, especially if the record is an outlier in a multidimensional space.

Liu et al., in 2008 at IEEE International conference [17], introduced a new algorithm with the name Isolation Forest. The main difference from other popular outlier detection methods is that Isolation Forest does not aim for profiling regular points, but rather, it goes directly to identifying anomalies/outliers explicitly. The Isolation Forest algorithm is using the power of combing multiple tree-based decisions in an ensemble method. The idea is simple and is based on randomly selecting a feature and then partitioning it by creating a random split value inside the range of the selected feature.

In principle, outliers have different characteristics in the feature space compared to other regular observations. This is because they tend to lie further away from the data cloud. Therefore, using a random partitioning approach as the one used by the isolation forest makes detecting outliers much easier as outliers will end up closer to the root of the tree and with fewer splits. When looking at it from a different angle, we will notice that outliers detected by isolation forest would, in general, have the shortest path towards the root node of the tree compared to regular data points [18].

The previous concept can be easily understood by looking into Figure 28, which shows how the isolation forest algorithm work in the normal vs. abnormal observation. The figure is divided into two-part. On the left, we can see the decision boundaries of the tree when detecting a normal point, and on the right, we can see an abnormal point identification boundary [19].

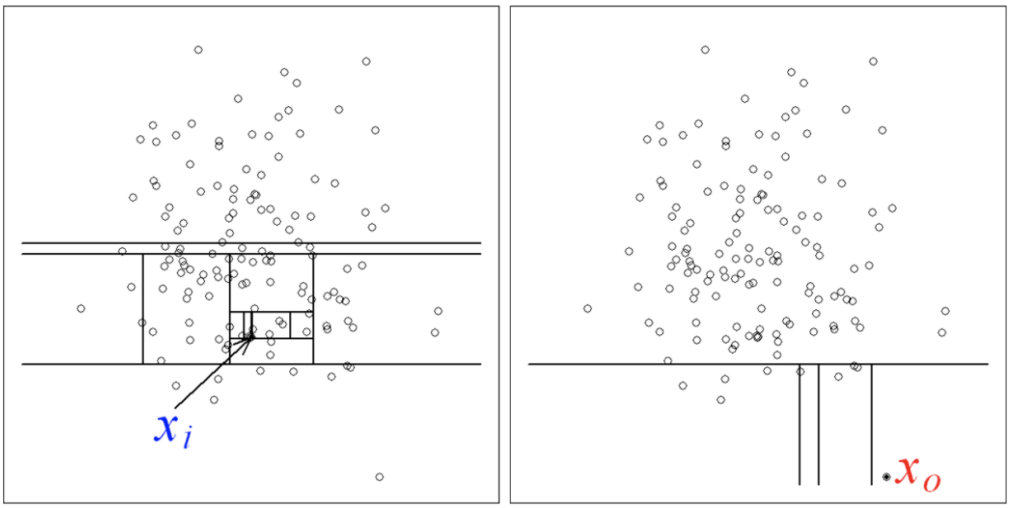
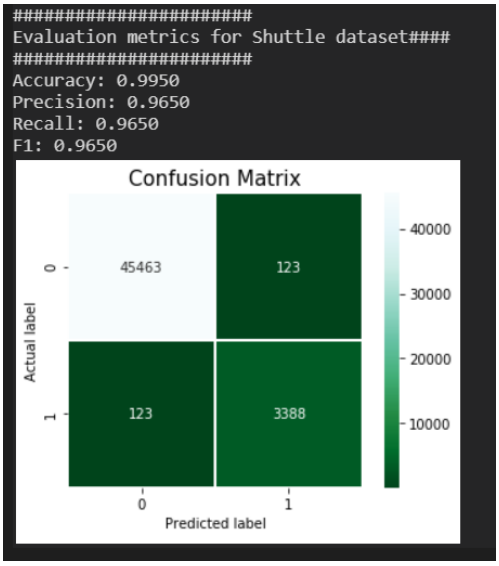


Figure 28 – *Isolation forest where the left panel shows the decision boundary of normal point and on the right panel we have boundary of an outlier observation, source* [18].

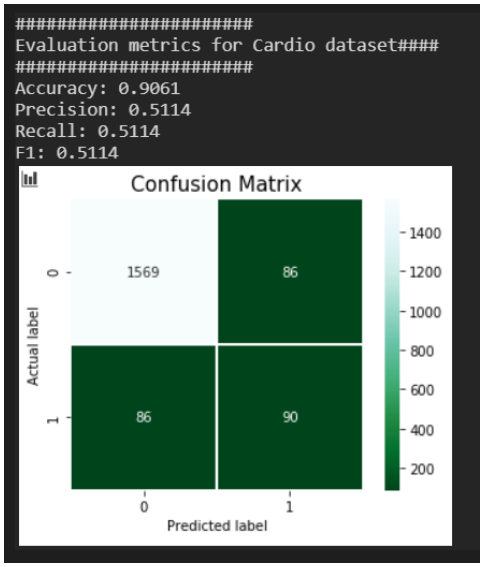
so now let us see a hands-on example. For simplicity, I will work with four datasets Shuttle, Cardio, Mammography and Annthyroid datasets which can be download from the Outlier Detection Datasets website (ODDS) [20]–[23]. The advantage of using datasets from this website is that outliers have been already identified and are added as a label inside the datasets. In this case, I can compare the performance of the isolation forest for predicting outliers compared to the ground truth of the dataset. The datasets contain (Shuttle: 49,097 observations; 8 Features; 7% missingness), (Cardio: 1,831 observations; 20 Features; 9.6% missingness), (Shuttle: 11,183 observations; 5 Features; 2.3% missingness), (Annthyroid: 7,200 observations; 5 Features; 7.4%). The observations are labeled, so we know which ones are anomalous upfront.

The isolated forest was able to correctly detect 96% of the outliers in the Shuttle dataset which is huge difference to the Mammography dataset which have the lowest detected outlier percentage with just 21.5% of outliers being labeled as outlier. This show us that the isolation forest is not always the best choice for detecting outliers in a given dataset. The dataset structure will have a big impact on the performance results of any machine learning outlier detection algorithm. Results of the four datasets is summarized in Figure 29.

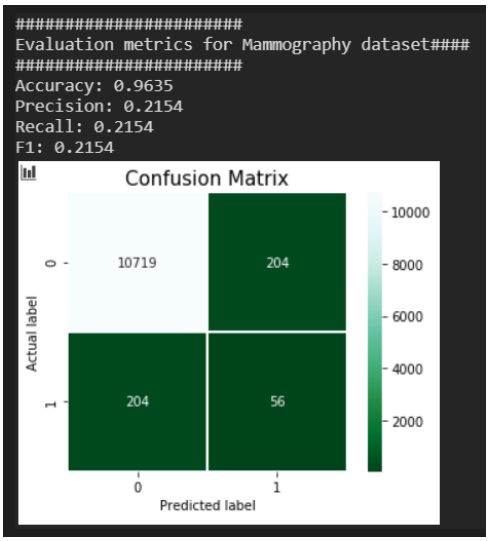
A)



B)



C)



D)

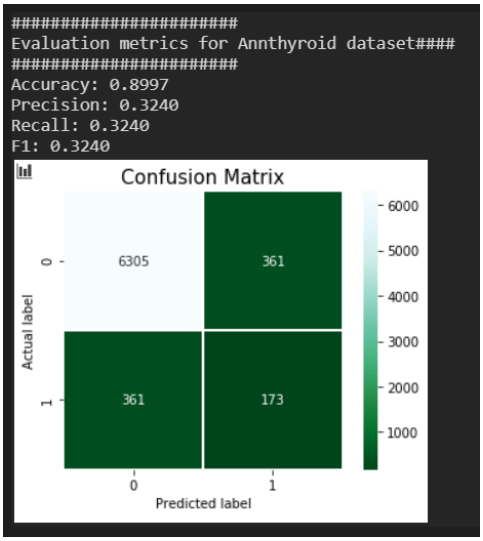


Figure 29 – *Confusion matrix for 4 datasets after performing isolation forest.*

Another advanced toolbox to be used in case the data scientist want to compare the performance of different outlier detection algorithms, would be the PyOD [24] library in python. PyOD was first published on the python PYPI in May 2018 and with the focus on the detection of outlying objects in multivariate data. The package has been downloaded 2.3 million time. PyOD includes more than 30 detection algorithms, from classical LOF (SIGMOD 2000) to the latest COPOD (ICDM 2020), an implementation of the isolation forest algorithm is included in this package as well. The library has been implemented in a very simple way which makes comparing and running multiple outlier detection algorithm very easy and less time consuming for the data scientist, that is why this package reached over 2 million downloads. Below figure show a subset of the algorithms and different techniques implemented by the PyOD library.

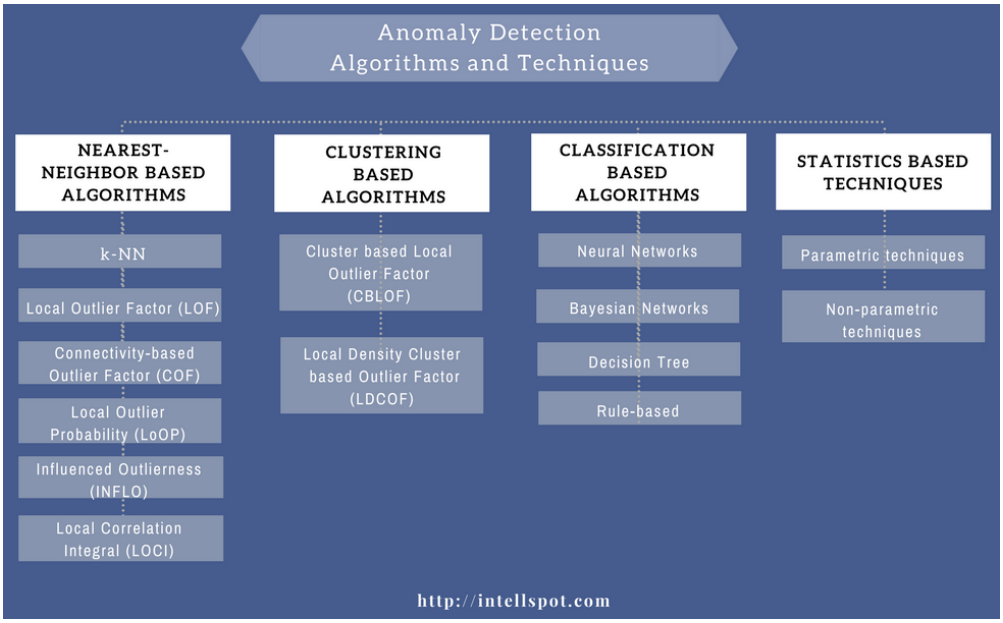


Figure 30 – *Outlier Detection aka Anomaly detection algorithms and techniques, source* [25].

I performed a fast comparison between 6 different algorithms from PYOD on the Mammography dataset. It was the dataset with the lowest recall using the isolation forest in the previous section analysis above. The results of this analysis are shown in the Table 2. However, we are still not getting any significant improvement over the isolation forest except for the Copula-Based Outlier Detection method, which achieved 50% improvement. This analysis's goal was not to have the best recall on the mammography dataset but to check how easy it would be to use multiple algorithms from the PYOD package. As expected, the implementation part went very fast and most of the algorithms exposed by the package contain default values, so not a lot of tunning was needed for the outlier detection algorithms.

Table 2 – *Comparison between 6 outlier detection algorithms from PYOD package on the Mammography dataset.*

Algorithm Name	***Accuracy***	***Precision***	***Recall***	***F1***
Angle-based Outlier Detector (ABOD)	0.9768	0	0	0
Cluster-based Local Outlier Factor (CBLOF)	0.9664	0.2769	0.2769	0.2769
Feature Bagging	0.9639	0.1697	0.1423	0.1548
Histogram-base Outlier Detection (HBOS)	0.9598	0.1289	0.1269	0.1279
K Nearest Neighbors (KNN)	0.9658	0.2325	0.2038	0.2172
PCA	0.9655	0.2577	0.2577	0.2577
Rotation-based Outlier Detection (RBOD)	0.9580	0.0962	0.0962	0.0962
Copula-Based Outlier Detection (COPOD)	0.9735	0.4308	0.4308	0.4308

2.5 Deduplication

The detection of two or more instances in an almost identical dataset or very high similarity is called deduplication. Such data duplication can occur due to multiple reasons as mentioned in [26]:

- Common spelling mistakes introduced when entering the data.
- Data is coming from different ERP systems with different standards for data entry.
- Changes in the customer information.
- One record has the customer's middle name, and the other record does not.
- Someone was intentionally making the change in the data to get some advantages or to be able to reinsert the same data twice for operation and support purposes.

Real-world data will not be a minor task to find duplicated records without using a fuzzy way to compare different features between two records. To find duplicates, usually, not all features will be used in the paired comparison. We would usually use names, addresses, phone numbers, dates, etc.

Table 3 – *Top 10 records from the restaurant dataset with the cluster column representing the duplicated clusters.*

	name	addr	city	phone	type	cluster
0	arnie morton's of chicago	435 s. la cienega blv.	los angeles	310/246-1501	american	0
1	arnie morton's of chicago	435 s. la cienega blvd.	los angeles	310-246-1501	steakhouses	0
2	arnie morton	435 s. la cienega boulevard	los angeles	310-246-1501	steakhouses	0
3	art's delicatessen	12224 ventura blvd.	studio city	818/762-1221	american	1
4	art's deli	12224 ventura blvd.	studio city	818-762-1221	delis	1
5	art's deli	12224 ventura blvd.	los angeles	818-762-1221	delis	1
6	hotel bel-air	701 stone canyon rd.	bel air	310/472-1211	californian	2
7	bel-air hotel	701 stone canyon rd.	bel air	310-472-1211	californian	2
8	bel-air	701 stone canyon road	bel air	(310) 472-1211	american	2
9	cafe bizou	14016 ventura blvd.	sherman oaks	818/788-3536	french	3

For performing a fuzzy comparison between two strings, we can use the jellyfish package, and for the fuzzy comparison between addresses, we can use the geocoder package.

Let us work with a real-life example from the famous restaurant dataset [27] used for deduplication exercises. This dataset consists of 864 restaurant records from Fodor's and Zagat's restaurant guides that contain 150 duplicates. It also includes the multiclass label column called cluster added to the dataset which is used to identify group of duplicated records. Table 3 show the top records in the restaurant dataset. I will be using the dedupe python package, which performs an active learning classification technique to find the duplicated pairs/cluster in our dataset.

You need to add labels for at least ten positive and ten negative duplicate cases using an interactive window as below figure. Before this, we need to choose which columns to be used. In this case, I will use the columns name, addr, postal, latlng. After selecting the list of columns to be used in the deduplication pairwise distance calculation, we will also need to decide which fuzzy search method to use for each column. We will use Jaro-Winkler for name, addr, and postal, while for the latlng, we will use the ExpLatLong as this field contain latitude and longitude data inside it; we will need to calculate distance.

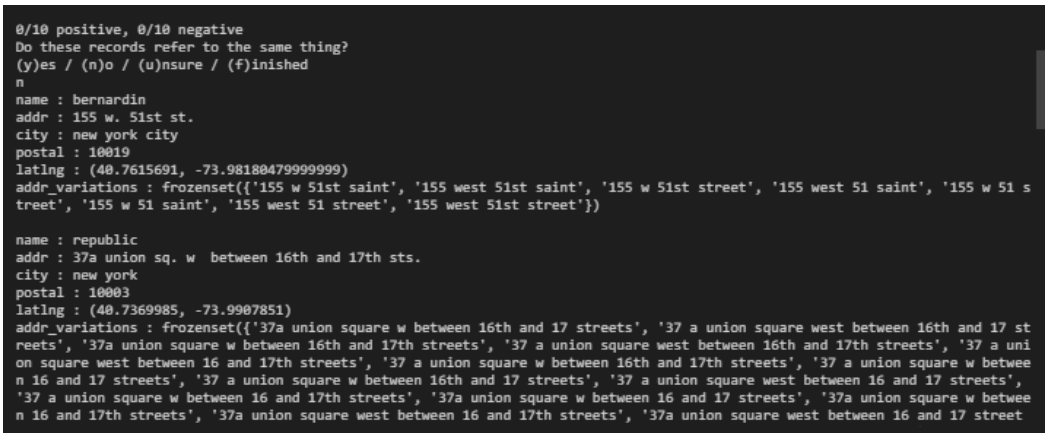


Figure 31 – *Active learning interactive console of dedupe python package.*

Based on our input dedupe start the analysis, and in a few minutes, depending on the data size, it produces an array with the clusters of duplicated records. As we have the original label, which tells which records are considered duplicates, we can calculate the number of true positives, false positives, and false negatives. Those numbers were 141, 3, and 9, respectively.

2.6 Categorical encoding

Categorical variables are those values that are selected from a group of variables that can take a limited set of values compared to a numeric variable are considered categorical variables. The predefined range of a categorical variable can be numeric or string data type. Usually, those values belong to a known finite set of categories or classes. The categorical variable can appear in any dataset as an ordinary predictor variable to be used in the model or it can be target variable for the model.

To use the categorical variable as a feature in a machine learning model, we need to first transform it from its space to a new space in a process known as categorical encoding. Encoding is becoming an essential step in the preprocessing of categorical data as all applications based on deep neural network learning would only accept inputs as numeric values.

A few procedures are available for encoding categorical variables for modeling; these procedures can be divided into three main categories as mentioned in [28].

1. **Classic Encoders:** well, known and widely used, for example, Ordinal, OneHot, Binary, Frequency, and Hashing encoders.

2. **Contrast Encoders:** another alternative for encoding is looking at different levels of features, for example, helmert and backward difference.

3. **Bayesian Encoders:** a more sophisticated method for encoding is by using target variable info in the encoding process, for example, Target, Leave One Out, Weight Of Evidence, James-Stein and M-estimator.

The best library, which contains almost all the possible categorical encoders, can be downloaded from the python package index (PyPI) repository under the name category_encoders [29]. This package was first release on Feb 2016 and it became so popular that it has reached almost five million downloads since then. It includes a set of scikit-learn-style transformers for encoding categorical variables into numeric by means of different techniques. All encoders are fully compatible with sklearn transformers so that they can be used in pipelines or in your existing scripts. Supported input formats include NumPy arrays and pandas DataFrames. If the cols parameter is not passed, all columns with object or pandas categorical data type will be encoded.

I will not go over an example of how to use this package as it is a straightforward task. Still, I need to say that this package will not choose the best encoder based on your dataset, so in this case, you need to select your encoders per column and then call the category_encoders different transformers on the corresponding features.

An interesting cheat sheet that can help you choose between the different encoders is shown in the figure hereinafter created by Feature Labs.

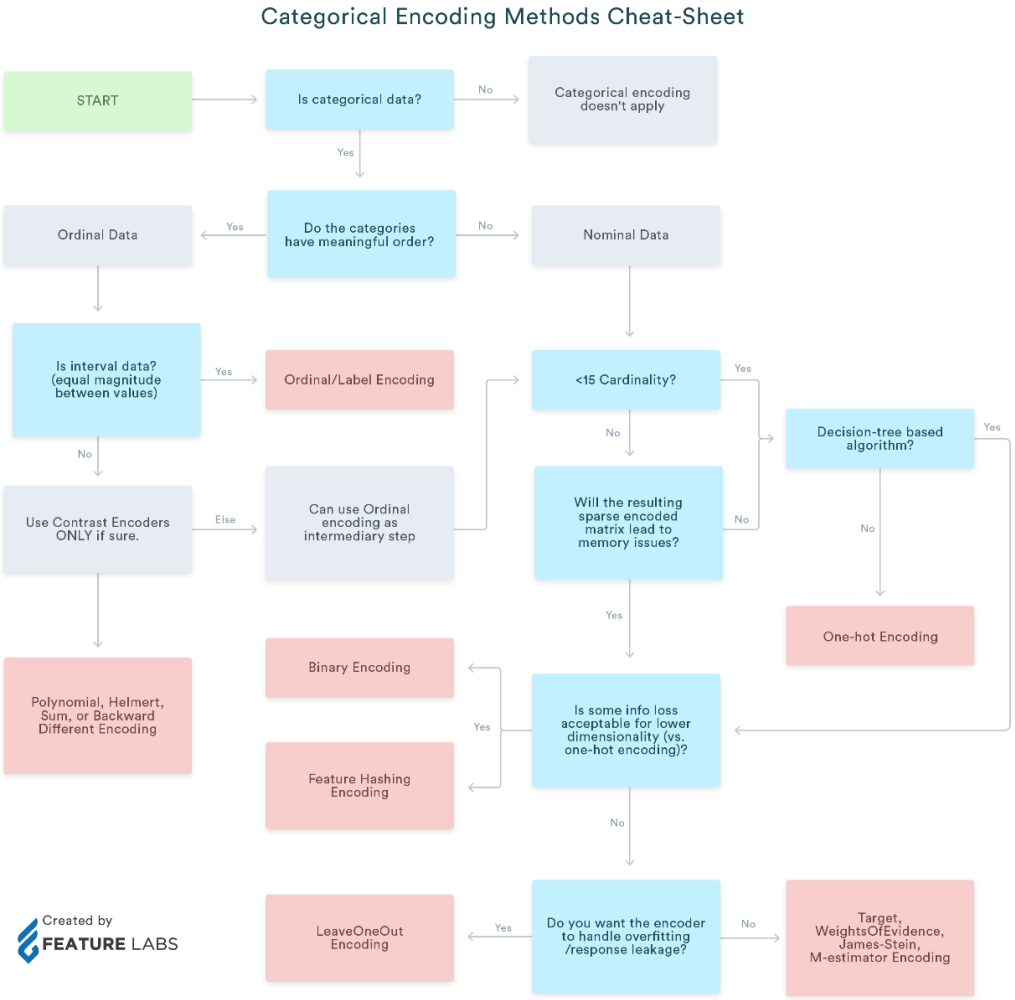


***Figure 32** – Categorical encoder methods Cheat-Sheet by Featurelabs* [30].

2.7 Feature transformation and scaling

As we had shown in the previous section, categorical features need encoding, the same principal would be required for numeric features. In this case, we would perform any or all the following: scaling, standardizing, and transformation. These steps are needed to treat skewed features and rescale them. The majority of the machine learning algorithms performance is susceptible to data quality. As the data quality goes up, so does the model performance. They go hand in hand in a direct positive correlation.

Statistical algorithms, in general, assume that features used in them to be normally distributed. This advantage of having normally distributed data is not just valuable for simple algorithms but also for deep learning and regression-type algorithms.

Depending on the distribution of the numeric features you are interested in, we would try to perform some transformation to make them follow normality as much as possible. A

statistical normality test such as the Shapiro-Wilk. Thus, it is not usually needed as our goal is to have a feature with a gaussian-like appearance in shape, and it does not have to be exactly normal. To accomplish the goal mentioned above, we would need to apply a mathematical function to each data point in the original distribution to get this new desired distribution.

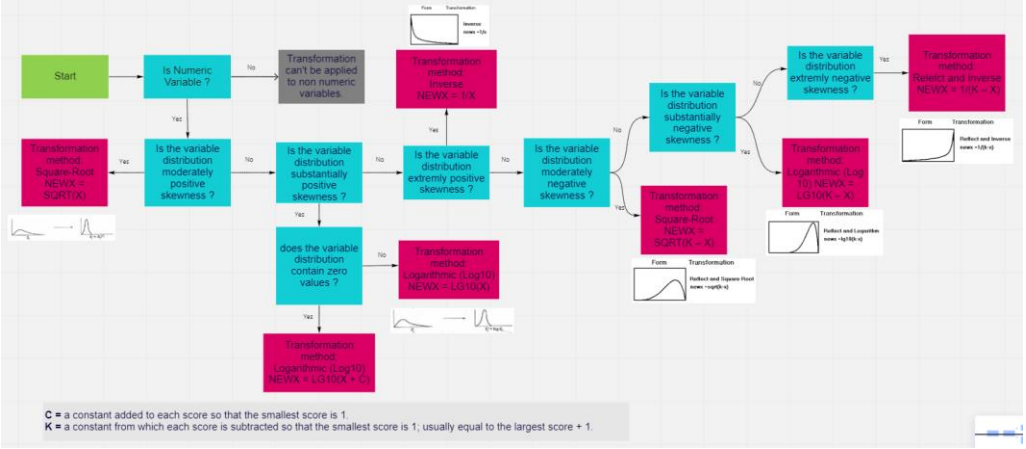


Figure 33 – *Feature transformation cheat sheet adapted from* [31] *and* [32].

Unfortunately, choosing the correct transformation for your variable is not an easy task to automate, but in general, you can choose between five major transformations as mentioned in [33], which are square root, cube root, logarithm, Cox-Box and Jeo-Johnson transformations. I tried to summarize the possible questions which we should answer before choosing the appropriate transformation based on our variable distribution, this summarization is shown in Figure 33. When choosing one of the mentioned transformations we need to remember that outliers can have a dramatic effect on the produced transformer. In a new paper from KUL [34] this effect of outlier had been accounted for and the code for this is available but only in R for the moment.

Some algorithms would need the features passed to it to be on the same scale, and this is often the case in algorithms that are based on the distance between records. To achieve this prerequisite task for these algorithms, we would need to use scaling techniques. Not all modeling algorithms would need scaling step, mainly Linear & Logistic Regression, KMeans/KNN, Neural Networks, PCA would need feature scaling before calling the algorithm. Entropy & Information Gain-based algorithms such as Tree-Based Algorithms, Decision Tree, Random Forest, Boosted Trees (GBM, light GBM, XGboost) would not need scaled features [33].

The four main scalers supported by the famous scikit learn package are: Minmax scaler, Robust scaler, Standard scaler, and Normalizer. A quick summary of when we can choose between these scalers is shown in Figure 34.

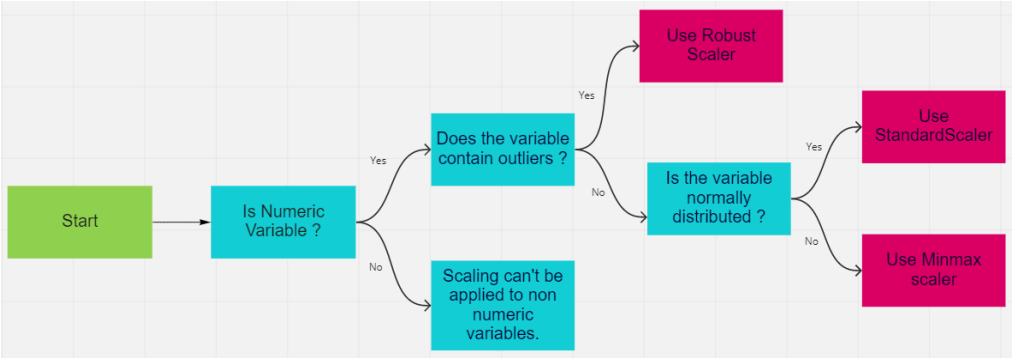


***Figure 34** – Numeric Feature scaling set of rules to choose the correct scaling method.*

2.8 Feature engineering

Finally, after cleaning and getting the feature ready for the analysis part, the last part would be feature engineering. This part is not considered a data cleaning part. However, it is still considered part of the preparation phase needed before using or comparing different machine learning methods. Having relevant features can usually significantly impact the performance of the use case more than the choice of the algorithm; because of this, getting those relevant features is an essential prerequisite to start the analysis phase. We will not have all the necessary features predefined, so we need to begin artificially synthesizing them. Feature engineering is the task of coming with new relevant features from already existing features. In the last five years, data scientists had access to new tools that can automate feature extraction. A comparison between those latest libraries is shown in Table 4 below. Next will highlight two of these libraries in the coming subsection.

***Table 4** – A comparison between different Feature engineering tools for python, source* [35].

Tools/Measures	Support for Type of Databases	Feature Engineering	Feature Selection	Open Source Implementation	Support for Time Series
Featuretools	Relational Tables	Yes	Yes	Yes	Yes
AutoFeat	Single Table	Yes	Yes	Yes	No
TSFresh	Single Table	Yes	Yes	Yes	Yes
FeatureSelector	Single Table	No	Yes	Yes	No
OneBM	Relational Tables	Yes	Yes	No	Yes
Cognito	Single Table	Yes	Yes	No	No

2.8.1 Featuretools

Let us start by introducing the Featuretools [36]. It was built as an open-source Python library based on a famous technique called deep feature synthesis. Featuretools is best used when we have a dataset compromising of a set of related tables. Starting from this as an input, the library will dramatically produce a new number of features. The library is relatively old, with the first version appearing in 2017, but it started to get more popularity recently, with over 50K downloads per month.

For a quick understanding of how the library work, I will go quickly over the steps needed to be able to call the relevant functions on your dataset, all the subsequent details are adopted from these two sources [37], [38]. First, we need to define all the different tables as entity objects in Featuretools. Then we need to define all the relationships between the tables as an entityset object. For this exercise, I am going to use just a small dataset for customer transactions which comes predefined inside the Featuretools and can be loaded using the "load_mock_customer" function. The structure of this dataset is shown in Figure 35 where we have all the different tables and their relationship to the customer table. The customer table would be our main table that we would like to enrich with new features, this will be shown soon later in this section.

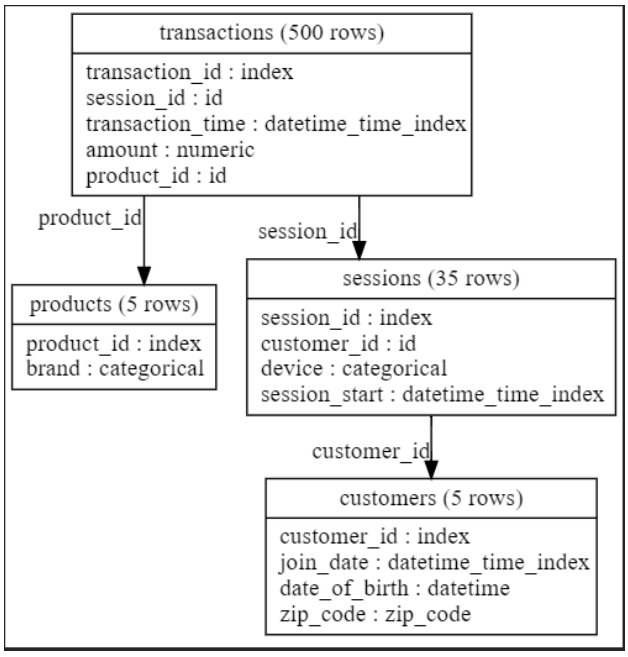


Figure 35 – *ER diagram of the Customer dataset.*

The second step in our process would be to define the feature primitive, which is a main component of the Featuretools library. The feature primitive is the instructions that you would give to the featuretools to help it create a new feature for you. Featuretools is trying to capture how data scientists use to create new features manually, and then based on this; It would start creating many advanced features. Usually, the feature primitive is based on simple calculations, and it can be divided into two categories:

- Aggregation: This is the process where we gather data in a certain way to produce new summary info about a given field. A famous example of such aggregation functions would be the mean, max, min, or standard deviation of a feature. Featuretools support aggregation across multiple tables based on the relationship between them, defined in the entity set. For the moment, the libraries have implemented 22 different primitives. The first ten primitives are shown in the following table.

Table 5 – *A list of the top 10 aggregation primitive/functions used by featuretools.*

	name	type	dask_compatible	koalas_compatible	description	valid_inputs	return_type
0	avg_time_between	aggregation	False	False	Computes the average number of seconds between consecutive events.	DatetimeTimeIndex	Numeric
1	max	aggregation	True	True	Calculates the highest value, ignoring `NaN` values.	Numeric	Numeric
2	mode	aggregation	False	False	Determines the most commonly repeated value.	Discrete	None
3	sum	aggregation	True	True	Calculates the total addition, ignoring `NaN`.	Numeric	Numeric
4	entropy	aggregation	False	False	Calculates the entropy for a categorical variable	Categorical	Numeric
5	trend	aggregation	False	False	Calculates the trend of a variable over time.	Numeric, DatetimeTimeIndex	Numeric
6	percent_true	aggregation	True	False	Determines the percent of `True` values.	Boolean	Numeric
7	median	aggregation	False	False	Determines the middlemost number in a list of values.	Numeric	Numeric
8	last	aggregation	False	False	Determines the last value in a list.	Variable	None
9	time_since_first	aggregation	False	False	Calculates the time elapsed since the first datetime (in seconds).	DatetimeTimeIndex	Numeric

- Transformation: In contrast to the aggregation function, which can be calculated based on information from multiple tables, the transformation function would be based on a single table, but one or more columns would be involved in creating the new feature. For example, extracting the year part from a DateTime field, or counting the number of days since a given event happening, or finding the difference between two columns if multiple columns will be used to construct the new feature. For the moment, the libraries have implemented 57 different transformations. The first ten primitives is shown in the table below.

Table 6 – *A list of the top 10 Transformation primitive/functions used by featuretools.*

	name	type	dask_compatible	koalas_compatible	description	valid_inputs	return_type
0	longitude	transform	False	False	Returns the second tuple value in a list of LatLong tuples.	LatLong	Numeric
1	day	transform	True	True	Determines the day of the month from a datetime.	Datetime	Ordinal
2	less_than_equal_to	transform	True	True	Determines if values in one list are less than or equal to another list.	Ordinal, Numeric, Datetime	Boolean
3	equal	transform	True	True	Determines if values in one list are equal to another list.	Variable	Boolean
4	percentile	transform	False	False	Determines the percentile rank for each value in a list.	Numeric	Numeric
5	time_since_previous	transform	False	False	Compute the time since the previous entry in a list.	DatetimeTimeIndex	Numeric
6	multiply_numeric	transform	True	True	Element-wise multiplication of two lists.	Boolean, Numeric	Numeric
7	greater_than_equal_to_scalar	transform	True	True	Determines if values are greater than or equal to a given scalar.	Ordinal, Numeric, Datetime	Boolean
8	greater_than_equal_to	transform	True	True	Determines if values in one list are greater than or equal to another list.	Ordinal, Numeric, Datetime	Boolean
9	negate	transform	True	True	Negates a numeric value.	Numeric	Numeric

To perform Deep Feature Synthesis (DFS) in featuretools, we use the dfs function passing to it an entity set, the target_entity (where we want to make the features), the agg_primitives to use, the trans_primitives to use, and the max_depth of the features. The max_depth parameter is used to tell how many levels of stacking of aggregated or transformed featured will be used. In my current example using the customer dataset, I will use the default aggregation and transformation functions, and I will use a max depth of 2. As there will be too much computation depending on the size of your dataset, the dfs function gives you the option to run in a dry mode if the feature_only is sent as True. The output of the function call, in this case, will be a list of the features without calculating it. You save a lot of computation time if you want to see the definitions of all the created features, and then based on this, you can exclude some of them, and then when you are happy with the feature list, you can rerun the function with feature_only to False. The featuretools can produce 148 new features, when using the customer table as the target table for the dfs function.

```
[<Feature: SUM(orders.MAX(order_products.total))>,
 <Feature: SUM(orders.MAX(order_products.unit_price))>,
 <Feature: SUM(orders.MEAN(order_products.quantity))>,
 <Feature: SUM(orders.MEAN(order_products.total))>,
 <Feature: SUM(orders.MEAN(order_products.unit_price))>,
 <Feature: SUM(orders.MIN(order_products.quantity))>,
 <Feature: SUM(orders.MIN(order_products.total))>,
 <Feature: SUM(orders.MIN(order_products.unit_price))>,
 <Feature: SUM(orders.NUM_UNIQUE(order_products.product_id))>,
 <Feature: SUM(orders.SKEW(order_products.quantity))>,
 <Feature: SUM(orders.SKEW(order_products.total))>,
 <Feature: SUM(orders.SKEW(order_products.unit_price))>,
 <Feature: SUM(orders.STD(order_products.quantity))>,
 <Feature: SUM(orders.STD(order_products.total))>,
 <Feature: SUM(orders.STD(order_products.unit_price))>,
 <Feature: MODE(order_products.orders.country)>,
 <Feature: MODE(order_products.orders.customer_name)>,
 <Feature: NUM_UNIQUE(order_products.orders.country)>,
 <Feature: NUM_UNIQUE(order_products.orders.customer_name)>,
 <Feature: PERCENT_TRUE(order_products.orders.cancelled)>]
```

Figure 36 – *Top list of the produced features based on 2 levels of aggregation and transformation of the original features.*

We used the default aggregations without thinking about which ones are "important" for the problem. We end up with many features, but they are probably not all relevant to the problem. Too many irrelevant features can decrease performance by drowning out the essential features. Finally, after generating the new features, we need to perform a new step for feature selection and feature importance to keep relevant features based on the problem in hand, which is not the main focus of this dissertation. This is possible still using the Featuretools package as it can also perform a round of feature selection on the new produced features.

2.8.2 AutoFeat

The next best feature engineering library when you have a single table is the Autofeat library in python [39]. This library can generate more than 3,500 features from just a table with 3 feature following a 3 steps of transformation and combination as shown in Figures 39 and 40. The library was released in January 2019 but it still didn’t get that much attention from the data scientist community with just 60K downloads so far.

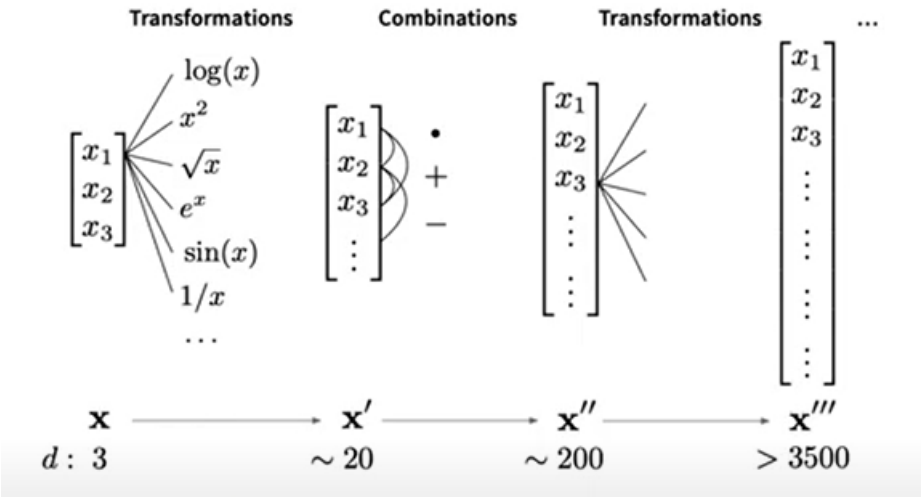


Figure 37 – *Generating new features based on transformation and combination of original features.*

AFR1	$1/x,\ x^3,\ x^2,\ \exp(x)$
AFR2	$\sqrt{x_1}/x_2,\ 1/(x_1x_2),\ x_1/x_2,\ x_1^3/x_2,\ x_1^2/x_2,\ \exp(x_1)\exp(x_2),\ \exp(x_1)/x_2,$ $\sqrt{x_1}\sqrt{x_2},\ \sqrt{x_1}x_2^3,\ x_1\log(x_2),\ \log(x_1)/x_2,\ x_1^3x_2^3,\ x_1^3x_2,\ x_1^3\log(x_2),\ \ldots$
AFR3	$x_1^3/x_2^3,\ \exp(\sqrt{x_1}-\sqrt{x_2}),\ 1/(x_1^3x_2^3),\ \sqrt{x_1x_2},\ 1/(x_1+x_2),\ x_1/x_2^2,$ $1/(\sqrt{x_1}-\log(x_2)),\ \lvert\sqrt{x_1}-\log(x_2)\rvert,\ \exp(\log(x_1)/x_2),\ \log(x_1)^2/x_2^2,$ $\lvert\log(x_1)+\log(x_2)\rvert,\ \ldots$

Figure 38 – *Example of the produced feature definitions after each of the three iterations.*

The idea is exceptionally straightforward; the input feature vector is changed using non-linear transformations (logarithmic, exponential, complementary, square root, etc.). Combining those changed or ordinary features to make more complex features and again apply non-linear changes. This can be repeated as numerous times as needed, but the added space grows expectedly rapidly. It is favored to halt after two or three cycles by which much of the features is assembled. Let us show how can this be done using a simple build-in dataset Boston [40] a dataset with 13 feature and 1 numeric target variable.

First, load your dataset in a pandas dataframe and split data for training and testing, then use AutoFeatRegressor or AutoFeatClassifier depending on the target variable type. In this case the target is a numeric variable so AutoFeatRegressor model is called (AutoFeatRegressor(verbose=1, feateng_steps=feateng_steps)) on the training data and then used to transform the test data, the detailed steps were adapted and tested based on [41]. The algorithm will perform feature selection as well on top of the produced features, in this case, for the Boston dataset, I tried multiple feature number of steps to perform in the feature engineering part. By using the default values of the

AutoFeatRegressor it produced a new DataFrame with 22 new features, the definition of the new features can be seen in the Figure 41. Then I quickly compare the effect of creating a simple linear regression model for prediction of the target variable, one time with the original features and another time using the extra new features produced by Autofeat. A big significant improvement on the R2 value of the regression is accomplished. The model before feature engineering will give R2 value of 0.7617 on the training data set and R2 of 0.6896 on the test dataset. After feature engineering we have a better R2 value of 0.9289 and 0.8566 on the train and test data, respectively. The comparison between original and predicted values can be seen in Figure 42.

```
df.columns
Index(['x000', 'x001', 'x002', 'x003', 'x004', 'x005', 'x006', 'x007', 'x008',
       'x009', 'x010', 'x011', 'x012',
       'x005**2/x010', 'x005**3/x009',
       'x000*x011**3', '1/(x007*x012)',
       'log(x004)/x007', 'x008**3*exp(x005)',
       'exp(x005)*log(x007)', 'x009**3/x012',
       'x006**2/x008', 'exp(x005)*log(x000)',
       '1/(x002*x007)', 'x012**3*exp(x005)',
       'sqrt(x000)*x004**3', 'x011/x010',
       'x000**2*x003', 'x009**3/x008',
       'x009**3*x012**2', 'x005**3*sqrt(x011)',
       'x006**3/x000', 'x009**2/x008',
       'x010*log(x005)', 'x006**3*x010**3'],
      dtype='object')
```

***Figure 39** – List of new produced features from Autofeat based on the original 13 features in the Boston dataset.*

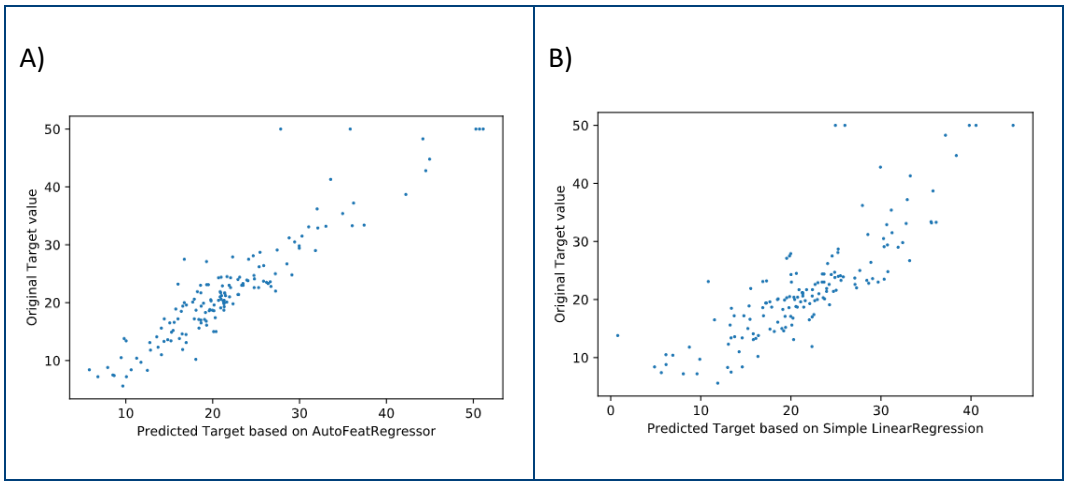


***Figure 40** – Scatterplot for the Boston dataset where in panel A we have the results of X axis results based on the original features and in panel B it is after feature engineering.*

2.9 Data drifting

Finally, the last part of the results section will be dedicated to the critical problem of model drifting which can happen to any model in production as mentioned in the introduction section before. In this last part I will quickly review a new package called pydrift which has been release in May 2020. This package will help in detecting drifting in the features used to create the current model. It will also check if the features relationship with the target variable still holds. I will use the titanic dataset to demonstrate the functionality of this package.

First, I will check the use of the DataDriftChecker class which is expecting two data frames to compare their distribution, If I split the dataset into training set and test set and send these as parameters to the constructor of the DataDriftChecker class then it will show no difference as the distribution between the two dataset are not significantly different. Once we stratify the original dataset using the sex feature into two new DataFrames which represent males survivors DataFrame and female survivors DataFrame, we will notice that when calling the DataDriftChecker classifier on these two new DataFrames that it will be able to see that the two datasets are different in the distribution of the sex variable, this is expected but it is just a simple example to show that DataDriftChecker will be able to detect such changes, the results is shown in the below figure. So based on Figure 43 we are able to see that the two DataFrames are different in the sex column distribution.

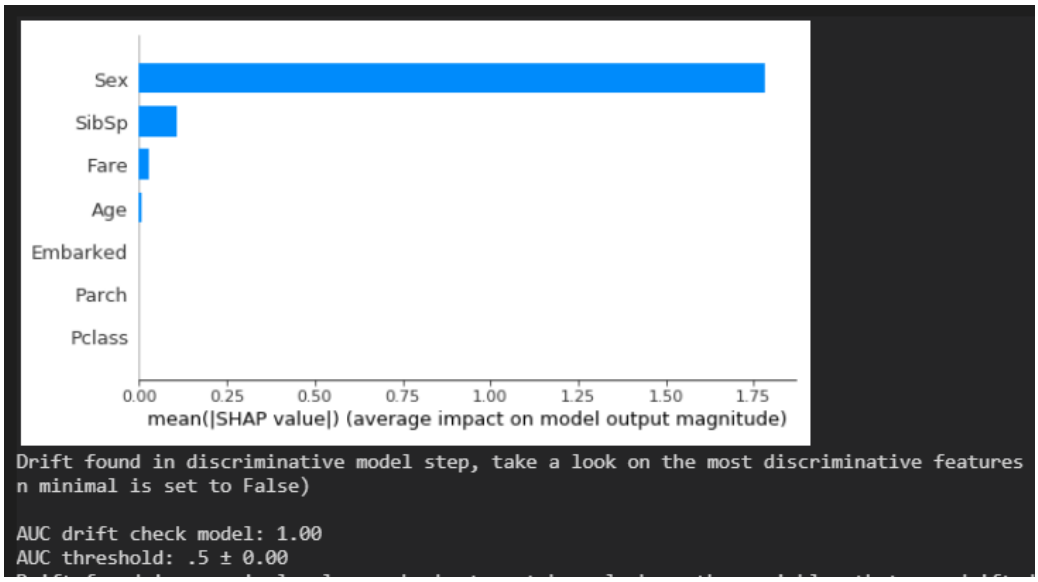


Figure 41 – *Most Discriminative Features plots based on CatBoostClassifier based on the sex feature discrimination.*

We can also divided our original dataset again into two dataset based on Pclass and Fare feature and check if the DataDriftChecker will be able to detect this, and again as expected we are able to detect which columns are different, results can be seen in Figure 44. The DataDriftChecker class contains a function called ml_model_can_discriminate which will call a CatBoostClassifier algorithm from sklearn to detect any difference between the two DataFrames, you can choice any algorithm from sklearn and send it to the discriminate function. You need to send an AUC threshold (default value is 0.5), if the model produce an AUC higher than the threshold this will mean that the two DataFrames drift from each other.

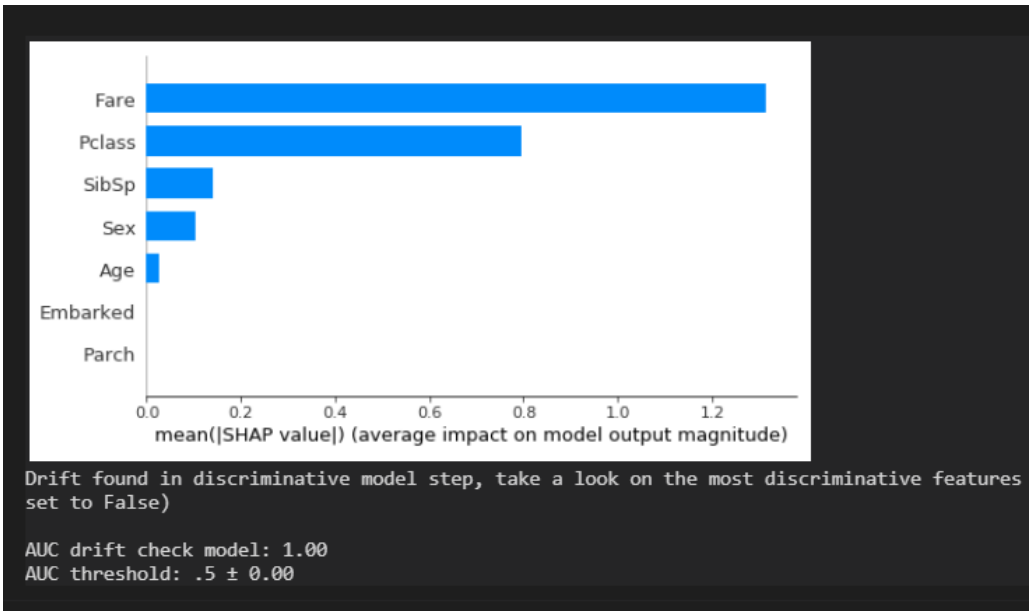


Figure 42 – *Most Discriminative Features plots based on CatBoostClassifier based on the Pclass and Fare features.*

3 Conclusion and recommendations

In this dissertation, my main objective was to gather more information about the latest automation package developed for python during the last five years. I was focusing more on this work to automate the different steps of the process and not on end-to-end automation. For the latter task, it would be harder to review at this moment as most of the tools that support end-to-end automation are not free tools. This review was more shifted to open-source python packages.

As I started to analyze more datasets for our clients as a step in our demo preparation routine, I felt the importance of speeding and automating these tasks as much as possible. The automation is not meant to remove the data scientist or data engineer from the loop but rather to give him more advanced tools and packages to be added to his arsenal to finish a demo for a client in a reasonable amount of time.

Once I had this feeling and urged inside me to find more packages to automate the data preparation and exploration phase, I was astonished to see how many open-source projects in the community exist. I think more packages had been developed during this pandemic than before. Maybe more data scientists and engineers had more free time to invest time in projects that would affect the whole data science projects life cycle time and quality.

List of Figures

List of Tables

List of References

[1] "What is Data Preparation? (+ How to Make It Easier) - Talend." https://www.talend.com/resources/what-is-data-preparation/ (accessed Jun. 08, 2021).

[2] "Databricks Delta Lake and Its Benefits." https://www.slideshare.net/databricks/databricks-delta-lake-and-its-benefits/10 (accessed Jun. 08, 2021).

[3] "Databricks Delta | element61." https://www.element61.be/en/competence/databricks-delta (accessed Jun. 11, 2021).

[4] "Organizations Have Low BI & Analytics Maturity | Gartner." https://www.gartner.com/en/newsroom/press-releases/2018-12-06-gartner-data-shows-87-percent-of-organizations-have-low-bi-and-analytics-maturity (accessed Jun. 05, 2021).

[5] "Data science." https://www.slideshare.net/RanjitNambisan/data-science-172551780 (accessed Jun. 08, 2021).

[6] "2018-gartner-magic-quadrant-for-data-integration-tools-july-2018.pdf.".

[7] G. L. Taboada, I. Seruca, C. Sousa, and Á. Pereira, "Exploratory data analysis and data envelopment analysis of construction and demolitionwaste management in the European economic area," *Sustain.*, vol. 12, no. 12, Jun. 2020, doi: 10.3390/su12124995.

[8] "mlops-infographic." https://azure.microsoft.com/mediahandler/files/resourcefiles/mlops-infographic/MLOps Infographic.pdf (accessed Jun. 07, 2021).

[9] "Automate Exploratory Data Analysis With These 10 Libraries." https://www.analyticsvidhya.com/blog/2021/04/top-python-libraries-to-automate-exploratory-data-analysis-in-2021/ (accessed Jun. 01, 2021).

[10] "dtale · PyPI." https://pypi.org/project/dtale/ (accessed Jun. 09, 2021).

[11] "sweetviz · PyPI." https://pypi.org/project/sweetviz/ (accessed Jun. 10, 2021).

[12] "autoviz · PyPI." https://pypi.org/project/autoviz/ (accessed Jun. 08, 2021).

[13] "A Guide to Handling Missing values in Python | Kaggle." https://www.kaggle.com/parulpandey/a-guide-to-handling-missing-values-in-python (accessed Jun. 09, 2021).

[14] D. J. Stekhoven and P. Bühlmann, “Missforest-Non-parametric missing value imputation for mixed-type data,” *Bioinformatics*, vol. 28, no. 1, pp. 112–118, Jan. 2012, doi: 10.1093/bioinformatics/btr597.

[15] “MissForest: The Best Missing Data Imputation Algorithm? | by Andre Ye | Towards Data Science.” https://towardsdatascience.com/missforest-the-best-missing-data-imputation-algorithm-4d01182aed3 (accessed Jun. 01, 2021).

[16] “GitHub - kearnz/autoimpute: Python package for Imputation Methods.” https://github.com/kearnz/autoimpute (accessed Jun. 10, 2021).

[17] X. Chun-Hui, S. Chen, B. Cong-Xiao, and L. Xing, “Anomaly Detection in Network Management System Based on Isolation Forest,” in *Proceedings - 2018 4th Annual International Conference on Network and Information Systems for Computers, ICNISC 2018*, Apr. 2018, pp. 56–60, doi: 10.1109/ICNISC.2018.00019.

[18] “Outlier Detection with Isolation Forest | by Eryk Lewinson | Towards Data Science.” https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e (accessed Jun. 01, 2021).

[19] “Outlier Detection — Theory, Visualizations, and Code | by Dimitris Effrosynidis | Towards Data Science.” https://towardsdatascience.com/outlier-detection-theory-visualizations-and-code-a4fd39de540c (accessed Jun. 01, 2021).

[20] “Shuttle dataset – ODDS.” http://odds.cs.stonybrook.edu/shuttle-dataset/ (accessed Jun. 09, 2021).

[21] “Cardiotocogrpahy dataset – ODDS.” http://odds.cs.stonybrook.edu/cardiotocogrpahy-dataset/ (accessed Jun. 09, 2021).

[22] “Mammography dataset – ODDS.” http://odds.cs.stonybrook.edu/mammography-dataset/ (accessed Jun. 09, 2021).

[23] “Annthyroid dataset – ODDS.” http://odds.cs.stonybrook.edu/annthyroid-dataset/ (accessed Jun. 09, 2021).

[24] “pyod · PyPI.” https://pypi.org/project/pyod/ (accessed Jun. 10, 2021).

[25] “Anomaly Detection Algorithms: in Data Mining (With Comparison).” http://www.intellspot.com/anomaly-detection-algorithms/ (accessed Jun. 10, 2021).

[26] “Finding fuzzy duplicates with pandas - Max Halford.” https://maxhalford.github.io/blog/transitive-duplicates/ (accessed Jun. 01, 2021).

[27] “Duplicate Detection, Record Linkage, and Identity Uncertainty: Datasets.” https://www.cs.utexas.edu/users/ml/riddle/data.html (accessed Jun. 10, 2021).

[28] “Tutorial-Encoding-Categorical-Variables | Kaggle.”

https://www.kaggle.com/paulrohan2020/tutorial-encoding-categorical-variables?scriptVersionId=51643274 (accessed Jun. 01, 2021).

[29] “category-encoders · PyPI.” https://pypi.org/project/category-encoders/ (accessed Jun. 11, 2021).

[30] “Open Source | Feature Labs.” https://www.featurelabs.com/open/ (accessed Jun. 08, 2021).

[31] “DATA TRANSFORMATION.” Accessed: Jun. 08, 2021. [Online]. Available: http://abacus.bates.edu/~ganderso/biology/bio270/homework_files/Data_Transformation.pdf.

[32] “Transforming Data for Normality - Statistics Solutions.” https://www.statisticssolutions.com/transforming-data-for-normality/ (accessed Jun. 11, 2021).

[33] “Transformation & Scaling of Numeric Features: Intuition | by Manu Sharma | Towards Data Science.” https://towardsdatascience.com/transformation-scaling-of-numeric-features-intuition-7f4436e8e074 (accessed Jun. 01, 2021).

[34] J. Raymaekers and P. J. Rousseeuw, “Transforming variables to central normality,” 2020.

[35] “The Best Feature Engineering Tools - neptune.ai.” https://neptune.ai/blog/feature-engineering-tools (accessed Jun. 01, 2021).

[36] “featuretools · PyPI.” https://pypi.org/project/featuretools/ (accessed Jun. 11, 2021).

[37] “Automated Feature Engineering Basics | Kaggle.” https://www.kaggle.com/willkoehrsen/automated-feature-engineering-basics (accessed Jun. 10, 2021).

[38] “Automated Feature Engineering via Featuretools - Cloud. Big Data. Analytics... and so on.” https://krinkere.github.io/krinkersite/featuretools.html (accessed Jun. 11, 2021).

[39] “autofeat · PyPI.” https://pypi.org/project/autofeat/#description (accessed Jun. 10, 2021).

[40] “Index of /ml/machine-learning-databases/housing.” https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (accessed Jun. 10, 2021).

[41] “Guide To Automatic Feature Engineering Using AutoFeat.” https://analyticsindiamag.com/guide-to-automatic-feature-engineering-using-autofeat/ (accessed Jun. 01, 2021).