Implementing Pipelines to reduce and improve python analysis

By Tajaldeep Singh Ahluwalia
Jacobs University

Table of contents

Executive summary

Machine learning has become a fundamental part of business in recent years as a means of solving business problems. Every now and again, a new algorithm is released that improves upon the present state of the art. It's only natural for organisations (of all kinds) to strive to incorporate machine learning's tremendous capabilities into their fundamental functions.

This paper looks into the process of deploying a machine learning application by using scikit learn pipelines. The process reduces the total lines of code that are required in a python analysis and provides a clean lineage for the data transformations. The process also helps the code to perform the analysis through lineage and inter-dependence between each step, hence making the code more readable and performant.

Task/goals

This paper presents a quick introduction to the area of machine learning. Unsupervised and supervised machine learning are the two main types of algorithms that dominate the data analysis world. We have a variable (label) we wish to forecast as accurately as possible. In order to extract insights from the data, we use a wide variety of methodologies without any particular aim. This helps in understanding the data that is provided. Unsupervised learning techniques include clustering, which is used for consumer segmentation.

We can further divide supervised tasks into regression problems (when the objective variable is a continuous number, such as income or the price of a house) and classification problems (where the target variable is a discrete number, such as the price of a house) (where the target is a class, either binary or multi-class). For our analysis, we look at the classification problem.

Data background

We address a binary classification problem in the financial business in this research. We use a dataset that was donated to the UCI Machine Learning Repository (a very popular data repository). The data is publicly available and comes form a reputed data source. The data is also relatively clean and contains general information that is captured by any financial institution. This helps in reproducibility of the analysis for any similar classification problem.

Approach

The purpose of the article is to predict who is likely to default by combining some basic information about consumers (such as gender, age, and education level) with their previous repayment history. The setting is as follows: estimate if the customer will default in October based on the prior 6 months of payments history.

The goal of this paper is to explore a real-world approach to a machine learning challenge, from data collection and cleaning to creating and modifying a classifier. We then use the pipelines to create the analysis with minimal amount of code.

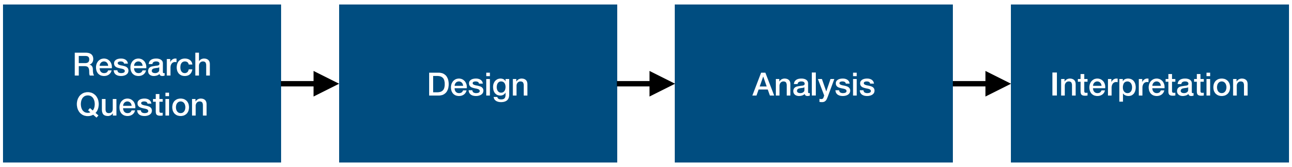
Results

The approach allows an easy deployment of the analytics pipeline that is able to predict the credit default. The analysis is able to present the clear visualisation of the important features and the results using plots and visual summaries. The pipeline allows the visualisation of the process and the steps that are carried out for the analysis.

Detailed report

Introduction

We explore the complete process of building an AI application to reflect the function and usefulness of statistics. Various procedures are required to experimentally explore a research question, as shown in the diagram below. See, for example, Weis and Ickstadt for more information on these steps (2018).



Starting with a detailed statement of the research question, the process progresses via a study design stage (including sample size planning and bias control) to a mathematical formulation (for example, as an optimisation problem) and numerical analysis. Finally, the findings must be deciphered.

AI frequently concentrates on the data analysis stage, while the other stages receive little or no attention. This can lead to key difficulties and perhaps misleading interpretations. There are common problems such as sampling bias or the use of ineffective analysis techniques that require assumptions that the chosen design does not meet.

Background of the data

The data for this paper was gathered from a bank. The study was prompted by the fact that, at the time, an increasing number of banks were providing willing clients with cash (and credit card) credit. Furthermore, regardless of their ability to repay, an increasing number of people have accrued huge sums of debt, resulting in defaults. The nature of the defaults are explored and relevant features are extracted in order to explain the nature of credit defaults.

Data Preprocessing

There are three data types: floats (floating-point numbers like 3.42), integers, and objects. The pandas representation of string variables is the last one. The number next to float and int denotes the number of bits this type employs to represent a value. The default memory allocation is 64 bits.

We also use a unique type known as category. The basic notion is that string variables are encoded as numbers, and pandas decodes them back into their original form using a custom mapping dictionary. When dealing with a small number of distinct string values, this is extremely handy (for example, certain levels of education, country of origin, and so on).

The total amount of the data can be decreased by 80% (memory-wise) using this simple change. We could also downcast the integer columns (which are now using the int64 type) to something smaller in memory, but we'd have to look at the min and max values for each column to see which type would be the best fit. In our case, we use the normal int64 type as the dataframe is not too large to be handled by any modern computer.

Exploratory Data Analysis

After loading the data, the next step is to perform exploratory data analysis (EDA). We learn about the data we'll be working with as a result of this. The following are some of the insights we aim to gather:

- What is the pattern of the variables' distribution?
  - Are there any outliers in the data, and if so, how should we handle them?
  - Is there a need for any transformations? Some models, for example, operate better (or require) normally distributed variables, hence approaches like log transformation may be useful.
  - Is there a difference in distribution by category (for example, gender or educational level)?
- Do we have any cases of data that is missing?
- How common are these, and which variables are they found in?
- Is there a linear relationship (correlation) between some variables?
- Is it possible to add additional features to the existing set of variables? Deriving hour/minute from a timestamp, day of the week from a date, and so on are examples.
- Are there any factors that we can eliminate from the study since they aren't relevant? A randomly generated client identifier is one example.

It is best to conduct univariate analysis (one feature at a time) on all important aspects to gain a thorough grasp of them before moving on to multivariate analysis. EDA is critical in all data science projects because it allows analysts to gain a better knowledge of the data, makes it easier to ask better questions, and makes it easier to choose modeling methodologies appropriate for the type of data being dealt with.

DESCRIBING THE DATAFRAME

We began the investigation with a very simple, yet powerful pandas DataFrame method—describe. It displayed summary statistics for all the numeric variables in the DataFrame, such as count, mean, min/max, and quartiles. We may deduce the value range of a specific attribute or whether the distribution was skewed by evaluating these measures (by looking at the difference between mean and median). We could also easily identify values that were outside of the realistic range, such as a negative or very small age.

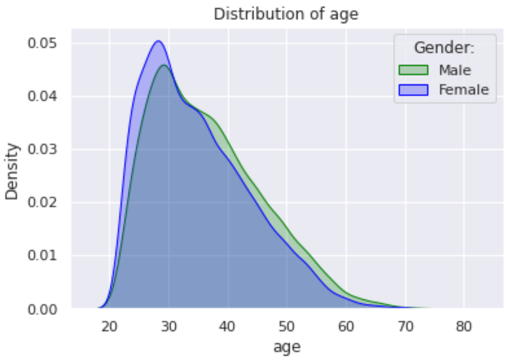
USING THE COUNT METRIC

Because the count metric reflects the amount of non-null observations, it can also be used to identify whether numeric characteristics have missing values. Running *df.isnull().sum()* is another technique to investigate the presence of missing values. Please check the recipe Dealing with missing values for additional information on missing values.

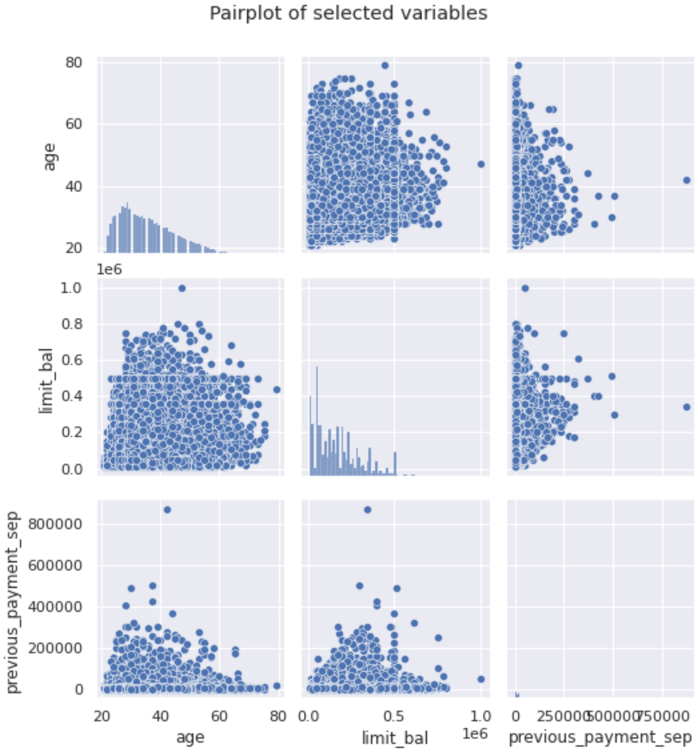
We added the *include='object'* option to the describe method in the third step to inspect the categorical features independently. The output differed from the numeric characteristics in that we could see the count, the number of distinct categories, which one was the most frequent, and how many times it appeared in the dataset.

EXAMINING DISTRIBUTIONS

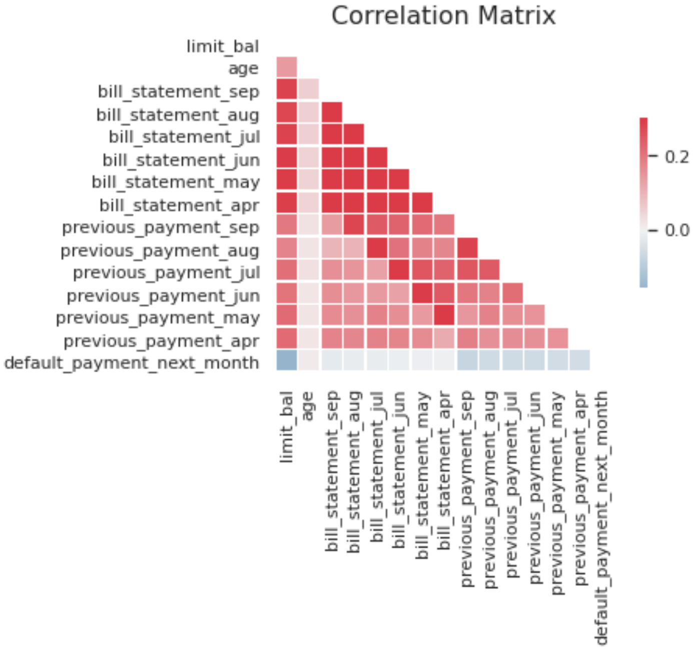
We then demonstrate a method for examining the distribution of a variable, in this case, the age of the clients. We did this by invoking *sns.distplot()* twice, each time with a different subset of the data (corresponding to age values per gender). We also had to use the *dropna* method to eliminate existing NA values.



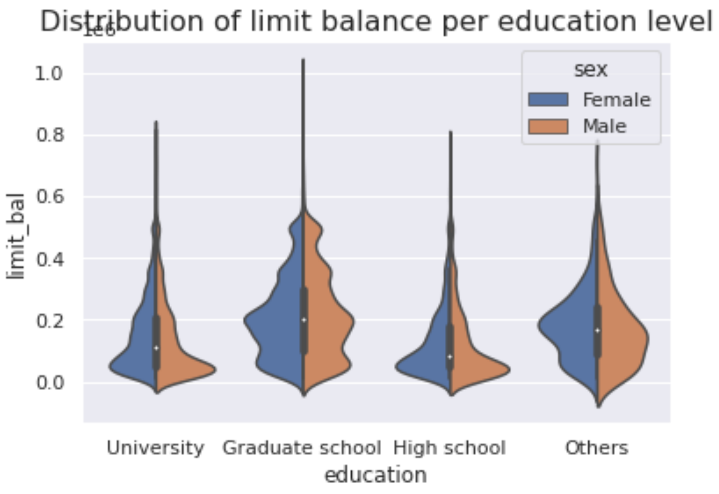
A pair-plot can be used to further this investigation. It generates a plot matrix, with the diagonal displaying the univariate histogram and the off-diagonal plots displaying scatterplots of two features. We may also see if there is a relationship between two attributes in this manner.



We then developed a function to generate a heatmap of the correlation matrix. We utilised a few of operations in the function to disguise the upper triangular matrix and the diagonal (all correlations equal to 1 on the diagonal). This made the output easier to understand.



Finally, we investigated the distribution of the limit balance characteristic by level of education and gender using violin plots. We made them with *sns.violinplot()*. The education level was designated as the *x argument*. We also set *hue='sex'* and *split=True*. Each half of the violin so represented a distinct gender.



Data analysis

ENCODING VARIABLES

The majority of machine learning algorithms only function with numerical data. That is why we must encode categorical information into a model-compatible representation. Some of the most common encoding methods are:

- Label encoding
- One-hot encoding

REMOVING OUTLIERS

LABEL ENCODING

In label encoding, the categorical value is replaced with a numeric value between 0 and the entire number of classes, if there are three separate classes, we use (0, 1, 2).

One potential difficulty with label encoding is that there is usually no association of any type between categories, but label encoding establishes one. This makes no sense if the categories are, say, countries. This can, however, work for features that indicate some form of order (ordinal variables), such as a service rating on a scale of Bad-Neutral-Good.

ONE-HOT ENCODING

We can utilise one-hot encoding to solve the preceding problem. In this approach, we generate a new column (often termed a dummy variable) with binary encoding for each feature category to indicate whether a specific row belongs to this category. One potential disadvantage of this strategy is that it significantly increases the dimensionality of the dataset (Curse of Dimensionality).

WHICH ENCODING

To summarise, we should avoid label encoding since it imparts misleading order into the data, which might lead to inaccurate conclusions. Decision trees, Random Forest, and other tree-based algorithms can function with categorical data and label encoding. However, one-hot encoding is the obvious representation for algorithms such as linear regression, models calculating distance metrics between features (k-means clustering, k-Nearest Neighbours, and so on), and Artificial Neural Networks (ANN).

MORE ENCODING OPTIONS

In addition to pandas and scikit-learn, we can use a package called *Category Encoders*. It is part of a series of packages that are compatible with *scikit-learn* and contains a variety of encoders that use a similar fit-transform technique. As a result, they can also be used in conjunction with *ColumnTransformer* and *Pipeline*.

IMPUTING MISSING VALUES

Most of the time, we do not work with clean, comprehensive data. One of the potential issues we'll have to deal with is missing values. We can classify missing values based on why they occur:

- **Missing completely at random (MCAR)—**The explanation for the missing data is unrelated to the remainder of the data. An example might be a respondent inadvertently skipping a question on a survey.
- **Missing at random (MAR)—**The missing values of the data can be deduced from data in another column (-s). For example, the absence of a response to a certain survey question can be decided conditionally by other characteristics such as gender, age, lifestyle, and so on.
- **Missing not at random (MNAR)—**When the missing data are not due to chance. People with really high earnings, for example, are often unwilling to share it.
- **Structurally missing data—**This is a subset of MNAR that is missing for a rational reason. For example, if a variable reflecting a spouse's age is absent, we can deduce that a specific person does not have a spouse.

Approaches that replace missing values with a single large value or the mean/median/mode are referred to as single imputation approaches since they replace missing values with a single specified value. There are, however, many imputation approaches, one of which being **Multiple Imputation by Chained Equations (MICE)**.

In short, the approach employs multiple regression models, with each missing value generated conditionally based on the non-missing data points. The decrease of bias produced by single imputation is one potential benefit of utilising an ML-based method to imputation. We use the MICE algorithm that can be accessed in *scikit-learn* library under the name *IterativeImputer* in the *impute* module.

DEFINING THE CLASSIFIER

A decision tree classifier is a straightforward yet crucial machine learning algorithm for both regression and classification issues. The term derives from the fact that the model generates a set of rules which can be seen as a tree when combined. By continually separating the features at a specific value, the decision trees separate the feature space into a number of smaller regions. To do this, they employ a greedy method (along with some heuristics) to locate a split that minimises the cumulative impurity of the child nodes (measured using the Gini impurity or entropy).

In the case of a binary classification problem, the algorithm attempts to create nodes that contain as many observations from one class as possible, hence minimising impurity. The prediction at a terminal node (leaf) is made on the basis of mode in classification issues and mean in regression tasks.

SCIKIT LEARN PIPELINES

A traditional analysis in python can be broken into many parts which are independent to each other. There is no clear lineage to the code and often times there is a risk that the code will break because of something that is modified in the previous steps. In a machine learning pipeline, this can be a critical point of failure.

The procedure necessitates the execution of several phases in a specific order, which might be difficult with frequent changes to the pipeline in the middle of the process. This is why Scikit-learn added Pipelines. We can use Pipelines to apply a series of transformations to data in a sequential manner, and then train a specified estimator (model).

One thing to keep in mind is that the Pipeline's intermediate phases must include the fit and transform methods (the final estimator only needs the fit method, though). Using pipelines has various advantages:

- The flow is much easier to read and comprehend—the sequence of operations to be performed on specified columns is obvious.
- The Pipeline enforces the sequence of steps.
- Reproducibility has been improved.

HOW THE PIPELINE IS IMPLEMENTED

We import the necessary libraries. The list may appear cumbersome, but this is because we need to merge several functions/classes used in earlier recipes.

The next step involves loading data from a CSV file, separating the target variable, and generating the stratified train-test split.

We then make two lists with the names of the numerical/categorical features—we'll apply various transformations based on the type of feature. We used the *select_dtypes* method to select the appropriate columns.

We begin preparing the individual Pipelines. For the numerical one, we merely intend to use the column median to impute the missing values of the characteristics. We provide a list of tuples containing the steps, each tuple containing the name of the step (for better identification) and the class we intended to use, in this instance the *SimpleImputer*.

We then create a Pipeline for categorical characteristics. This time, however, we chain two distinct operations: the *imputer* (which uses the most often occurring value) and the one-hot encoder. We also specify a list of lists named cat list for the encoder, in which we list all the available categories based on X train.

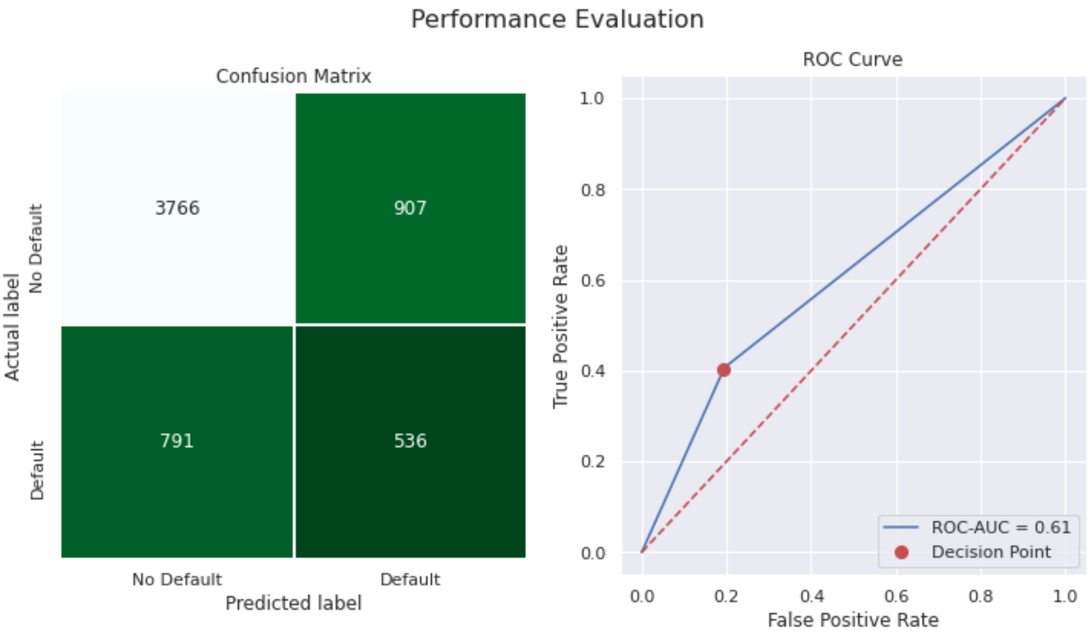
We define the *ColumnTransformer* object and use it to change the data in the columns. We again give a list of tuples, each containing a name, one of the previously established Pipelines, and a list of columns to which the modifications should be applied. We also specify *remainder='drop'* to remove any additional columns that had no modifications performed to them. Because the changes were done to all features in this situation, no columns were lost.

Finally, we utilise *Pipeline* once more to connect the *preprocessor* (the previously defined *ColumnTransformer* object) to the decision tree classifier (for comparability, we set the random state to 42). The final two steps are fitting the whole Pipeline to the data and measuring the model's performance with the custom function. The *performance_evaluation_report* function is designed to work with any estimator or Pipeline that uses the predict and *predict_proba* methods to generate predictions.

Results

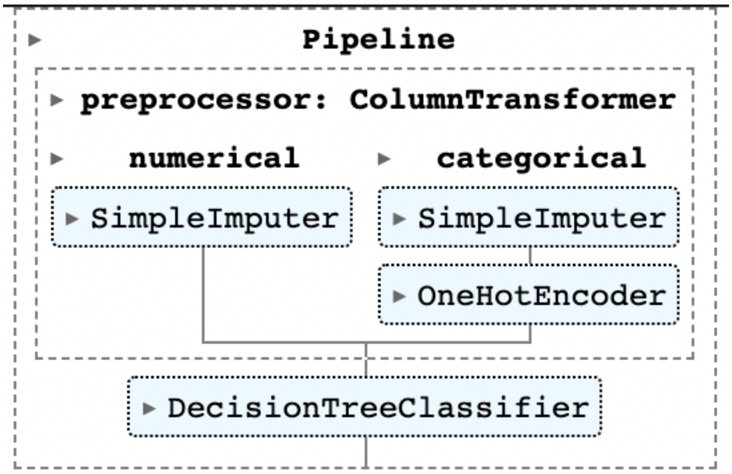
The results look promising as we have been able to implement the entire pipeline for the analysis in only a few lines of code.

With defining the functions such as *performance_evaluation_report* we are able to reuse it for any algorithm and get the summary for the performance. This evaluation metric allows the users to understand the model performance and also look at the decision point clearly.

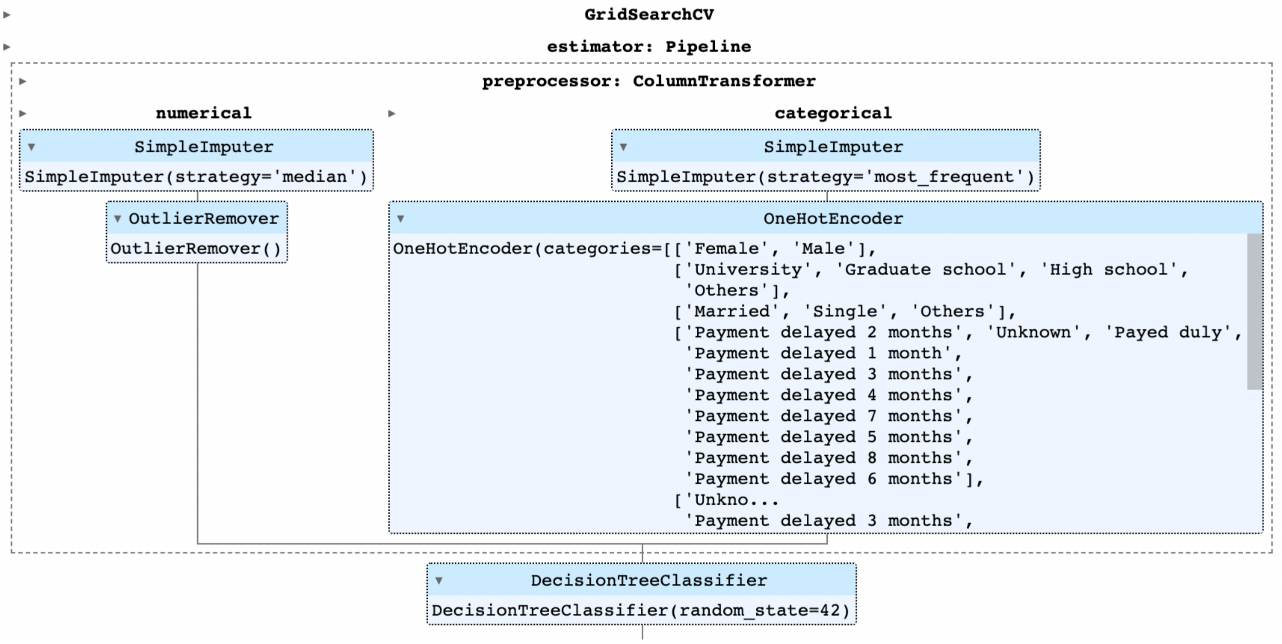


After we have implemented the pipeline, it is also essential to have a visual representation of how the pipeline is actually designed and functioning. In a traditional analysis, this can be a lengthily process as there is no demarkation for each steps. It is true that the code can be commented to increase readability. However, this is still limited to parsing though the entire code, which can be cumbersome.

To avoid this problem, we can use the *pipeline.fit(x,y)* method in the code and it yields an interactive infographic for the entire pipeline as shown below.



We also use the GridSearchCV classification in this paper. The results for the entire pipeline when expanded reveals a much more comprehensive overview of the analysis.



```
GridSearchCV
estimator: Pipeline
preprocessor: ColumnTransformer
numerical
SimpleImputer
SimpleImputer(strategy='median')
OutlierRemover
OutlierRemover()
categorical
SimpleImputer
SimpleImputer(strategy='most_frequent')
OneHotEncoder
OneHotEncoder(categories=[['Female', 'Male'],
                          ['University', 'Graduate school', 'High school',
                           'Others'],
                          ['Married', 'Single', 'Others'],
                          ['Payment delayed 2 months', 'Unknown', 'Payed duly',
                           'Payment delayed 1 month',
                           'Payment delayed 3 months',
                           'Payment delayed 4 months',
                           'Payment delayed 7 months',
                           'Payment delayed 5 months',
                           'Payment delayed 8 months',
                           'Payment delayed 6 months'],
                          ['Unkno...
                           'Payment delayed 3 months',
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

conclusion

We can conclude with the paper presented that implementing pipelines is a far superior approach to machine learning project. Traditional code is cumbersome and can lead to many technical difficulties if there is a change in the process and inputs at any stage of the analysis.

Pipelines allow a clear representation of the analysis that is more intuitive and interpretable. It is also helpful in reducing the amount of code that is required in a traditional analysis. Therefore, we can conclude that pipelines is a novel approach to tackle the drawbacks of traditional analysis.

Appendix

The code can be found on GitHub using the following link: https://github.com/tajalahluwalia/Data-Mining

The repository can be forked and any suggestions and changes are welcomed.

References

1. Friedrich, S., Antes, G., Behr, S., Binder, H., Brannath, W., Dumpert, F., ... & Friede, T. (2021). Is there a role for statistics in artificial intelligence?. Advances in Data Analysis and Classification, 1-24.
2. Weihs, C., & Ickstadt, K. (2018). Data science: the impact of statistics. International Journal of Data Science and Analytics, 6(3), 189-194.
3. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE, 104(1), 148-175.
4. Bisong, E. (2019). Introduction to Scikit-learn. In Building machine learning and deep learning models on Google cloud platform (pp. 215-229). Apress, Berkeley, CA.
5. Jolly, K. (2018). Machine Learning with scikit-learn Quick Start Guide: Classification, regression, and clustering techniques in Python. Packt Publishing Ltd.
6. Lewinson, E. (2020). Python for Finance Cookbook: Over 50 recipes for applying modern Python libraries to financial data analysis. Packt Publishing Ltd.
7. McKinney, W. (2012). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.".
8. McKinney, W. (2011). pandas: a foundational Python library for data analysis and statistics. Python for high performance and scientific computing, 14(9), 1-9.
9. Subasi, A. (2020). Practical Machine Learning for Data Analysis Using Python. Academic Press.
10. Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of machine learning research, 13(2).
11. Bergstra, J., Yamins, D., & Cox, D. D. (2013, June). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in science conference (Vol. 13, p. 20).
12. Embarak, D. O., Embarak, & Karkal. (2018). Data analysis and visualization using python. Berkeley, CA, USA: Apress.
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.