Master's Degree Program in Data Science and Advanced Analytics

Machine Learning

MACHINE LEARNING PROJECT REPORT

Group 28

Alex Morales Santander	20220658
Mohamed Ettaher Ben Slama	20221039
Skander Chaabini	20221041
Karim Miladi	20220720
José Pedro Lukoki	20221021

Academic Year

2022/2023

NOVA Information Management School

Universidade NOVA de Lisboa

Table of Content

ABSTRACT

An incipient discovered disease in England by Dr. Smith, called the 'Smith parasite' or SP, has already affected more than five thousand people, with no obvious connection between them. It can include symptoms such as fever or tiredness or being asymptomatic. This made it more arduous to detect clear patterns and transmission conditions.

The purpose of this project is to predict whether a patient is prone to suffer from SP or not.

The data provided has been divided into Demographic, Health, and Habits-cognate data. Additionally, this data has been split in training data and test data. Furthermore, three respective test datasets to evaluate the results with unseen data.

The goal of this work was to obtain a model to predict the disease with a high accuracy and to be resistant to noise in newly introduced data.

So, as a first step, the data was explored and understood. Then, to ascertain the good quality of the data, preprocessing was implemented. This step made the base for the modeling part in which several models were discussed. Determinately, it was concluded that the random forest was the best fit for the datasets and the most stable one.

Keywords: Data preprocessing , Supervised learning, Random Forest, Regularization, Jitter

Introduction

In England, Dr. Smith discovered the SP and it has already affected five thousand people. This strange illness has no clear symptoms which makes it harder to identify.

This report will explain the work done to build a predictive model that classifies sick people and healthy ones.

The datasets dealt with in this work contained Sociodemographic ,Health-related and Habits-related information containing features such as: birth year, drinking habits, exercise etc. Finally, the disease feature was also provided as a target variable with (Disease = 1) if the patient is sick or (Disease = 0) if not.

The ultimate goal of this investigation is to predict if the patient is more susceptible to suffer the SP. Machine learning can play a vital role in this process by helping to identify trends in the data that might not be apparent to human analysis alone.

Yet to do so the 800 rows of data must be explored and cleaned to build consistent stable models with reliable results to test on the 225 rows in the test data at first and any new unseen data in the future.

Data Exploration

It is very essential to first comprehend the data and get as many insights as possible before preprocessing and testing models. Data quality, correlations, and patterns between the variables should be examined and visualized, as well as identifying anomalies in the data.

1.1 Data Review

The data is contained in six different excel files presenting sociodemographic, health-related and habits-related information, for train and test data.

The first step was to merge the files into two datasets, one setting the training set and a second one for testing on unseen data using the patient ID column as an index.

As a result, the training dataset contains 800 rows and 18 columns including the target variable: Disease, and the test set contains 255 rows and 17 columns lacking the target column.

1.2 Data Analysis Exploration

After the datasets were merged, it was time to explore the data and fix its anomalies. The first step was taking an overall look at the datasets by displaying the head of the train and test data.

The balance of the data was checked by counting the proportion of the records having 0 or 1 in the disease variable. It was observed that the data is balanced with 51.375% sick records and 49.625% non-sick records.

After that, a data info check was performed. As shown in **Table 1**, it was discovered that 13 missing values exist in the education column. The types of the variables were also displayed, making the distinction between categorical variables and numerical variables clearer .

The next step was extracting the categorical features and the numerical features apart from the original training data, since the test for later steps is not the same for them.

Categorical features have been displayed in count plots, showing the distributions of each one of the nine categories. The count plots are shown in **Figure 1** to **8** (Annexes)

For the eight numerical variables, a histogram was plotted for each variable to observe its distribution. The Histograms are shown in **Figure 9** (Annexes)

The previous steps helped in the data preprocessing part and put the path to how to deal with the categorical and numerical variables.

It was discovered in the ‘region’ feature that London existed twice in uppercase and lowercase.

Finally, a check for duplicated rows was executed, followed by a check for duplicate Patient IDs. No duplicates were found in rows nor in Patient IDs.

1.3 Coherence check

A duplicate check was performed on the patient names' values. There were duplicated names in the dataset, but it was decided to keep both rows as it was a coincidence of having two patients with the same name.

Also, two variables (mental health and physical health) were calculated using the number of days in which a patient felt ill in the last 30 days, a check was made to ensure that no values exceeded 30 in these two variables. As a result, no record had incoherent values.

1.4 Outliers Check

The next step was performing an outliers check. Outliers are data points that fall significantly outside the range of the rest of the data. They can be caused by measurement errors or errors in data entry,

or they may represent genuinely unusual observations. Outliers can enormously affect the model's performance. Thus, the need to deal with them carefully.

For this matter, box plots were created to identify points outside the whiskers and assess the situation. More explanation is present in **figure 10** (Annexes)

After examining the box plots, some research was made to evaluate whether the points that seem to be outliers are typos or possible extreme values.The boxplots revealed the presence of several outliers that will be dealt with in the outliers handling section.

In summary, after exploring the data, it can be said that it is a balanced dataset without any duplicated rows, only 13 missing values in one column (Education) and that it has some outliers that need to be dealt with .

Data Preprocessing

Cleaning the data and its preparation for modeling is the purpose for the data preprocessing step. This might include filling data gaps from the data exploratory analysis step and dealing with missing values, identifying the important features, transforming the data, and creating new relevant features when it is best suited. This is a crucial phase, as the model's accuracy will depend on the quality of data that is being put into the algorithm to predict if the patient has a disease or not .

2.1 Outliers treatment

As mentioned in the first part, there are outliers to deal with. However, It is essential to note that determining whether a value is an outlier that should be removed or not is very subjective. While there are certainly valid reasons for throwing away outliers if they are the result of a computer glitch or a human error, eliminating every extreme value is not always a good idea. In this case, two methods were used to deal with outliers, the IQR method and mean standard deviation method.

The IQR method sets up a "fence" outside of Q1 and Q3. Any values that fall outside of this fence are considered outliers. To build this fence it takes 1.5 times the IQR and then subtract this value from Q1 and add this value to Q3 .For instance, some outliers are unusual values and impossible to occur.

Mean and standard deviation method calculate and compare the mean and standard deviation of the residuals. If a value is a certain number of standard deviations away from the mean, that data point is identified as an outlier. The specified number of standard deviations is called the threshold. The default value is 3.

In the outliers elimination process, IQR and Mean Standard Deviation methods were calculated for each numerical feature. Each method retrieved the number of outliers that would be deleted from the dataset. The process of which method would be chosen was decided using the accuracy of the models combining these methods. As a result the quantity of rows dropped is 15 and the percentage of rows dropped is 1.88% .

2.2 Data Split

This step consists of a separation of the training dataset into training and validation. The training dataset will be mixed and randomly split into a training set composed of 70% of the data, and a validation set, representing 30% of the data.

As a result we obtained one dataset for training which contains 549 rows and 17 columns and another one for validation with 236 rows and also 17 columns.

2.3 Feature engineering

The output of any ML algorithm predominantly depends on the quality of input being passed. The process of creating appropriate data features by applying the problem context is called feature engineering, and it is one of the most important aspects of building an efficient ML system.

The output of feature engineering is a clean and meaningful set of features that can be used by algorithms to identify patterns and build an ML model, which can further be applied on unseen data to predict the possible outcome.

Due to city names inconsistencies in the region variable it was decided to put all values in lowercase.

Three new features have been created: Age, BMI and Gender.

The Age feature replaces the Birth Year feature subtracting the current year with the birth year value.

The feature “BMI” was also created using “Weight” and “Height” in training, validation and test sets and then dropping the two features.

The feature “Gender” was also created using the Name feature as a base. Separating the name and the ‘Mr’ and ‘Mrs’.

The values greater than 100 in the feature “Age” were replaced with NaNs and then the nan values were filled with the mean of the “Age”.

All these changes were made in training, validation and test dataset.

After applying the above changes, a check of the distributions of target variables across all categorical variables was made using bar plots (see **figures 13** to **22**).

2.4 Scaling

Previously to any scaling process, the datasets were separated into categorical and numerical data.

For this dataset, we tried four different approaches:

- MinMax scaler
- Standard scaler
- Robust scaler

After running all approaches , the best score for scaling was Robust Scaler.In fact, Robust scaler scale features using statistics that are robust to outliers.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

The scaling was applied to all numerical data, that includes training, validation and test datasets.

2.5 Feature Selection

Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable.

It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

2.5.1 Numerical features

Filter Methods:

- Univariate selection: Univariate feature selection examines each feature individually to determine the strength of the relationship of the feature with the response variable. For the current data, all the features that have a variance value greater than 0 are selected. All the features were selected.
- Spearman Correlation: Spearman Correlation is a non-parametric test used to measure the degree of association between two variables. If two features are highly correlated, then they are redundant and would be an obstacle considering our model training.

From the correlation plots , there is no strong correlation so no decision can be made regarding feature selection in this step using correlation (see **figure 11)**.

Embedded methods:

Embedded methods are feature selection methods that include building a model capable of understanding which features are more important than others. Three methods have been used:

- Lasso Regression (see **figure 12**)
- Sequential Forward Selection (SFS)
- Sequential Backward Selection (SBS)

Wrapper Methods:

Wrapper methods are an iterative process with different subsets of features being selected and evaluated each time, until the optimal subset of features is found.

- Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. The result of the rfe model and the features selected are “'High_Cholesterol', 'Mental_Health', 'Physical_Health'

After using the wrapper, embedded and filter methods , the numerical features selected are :

- 'High_Cholesterol'
- 'Mental_Health'
- 'Physical_Health'

2.5.2 Categorical variables

For the categorical variables selection, Chi-square method was used, it measures the degree of association between two variables. After using chi-square, the categorical features selected are:

- Drinking_Habit
- Exercise
- Fruit habit
- Checkup
- Diabetes

Also, a visual representation of the data helped to understand the distribution of the dependent variable in each feature and select those that have an unbalanced distribution.

After finishing feature selection, then encoding is proceeded which is basically converting categorical variables to numerical

Modeling

In this phase, different models were applied in order to predict whether a certain patient has a disease (1) or not (0). Since the target variable is binary, classification algorithms were opted for to solve this problem. The applied classifiers are the following:

- Decision Tree Classifier
- Gaussian Naive Bayes
- Random Forest Classifier
- Bagging Classifier
- Extra Trees Classifier
- Histogram Gradient Boosting Classifier
- Logistic Regression
- Gradient Boosting Classifier
- K Neighbors Classifier
- Support Vector Machine Classifier
- Multi-layer Perceptron Classifier
- AdaBoost Classifier
- Stochastic Gradient Descent Classifier
- Voting Classifier
- Stacking Classifier

The scores measurements of all these classifiers with default parameters are found in **Figure 23** (Annexes)

3.1 Hyper-parameters Settings

The first approach was to run all the models with their default parameters. As a second method, hyperparameter tuning was applied, trying different sets of parameters with each algorithm in order to improve their accuracy.

To obtain the best hyper parameters, the task was divided in two sections:

- Randomized Search (RS)
- Grid Search (GS)

Using Grid search, in the first instance, will result in an extensive computational and time consuming task. For this reason, using RS can shorten the time and resources spent in hyperparameter tuning. RS was runned 500 times using a grid of continuous values for numerical hyper-parameters and for categorical parameters were used all the available options for each one of those.

This step narrows the area of search and GS can be performed reducing the time and resources used in a considerable way.

Both, RS and GS were fitted to the train and validation data in order to compare the accuracy of these models with the model in its native state. A dataframe containing the performance measurements using the default, RS, and GR parameters for each implemented model can be found (Check exemple

Figure 24). Finally, a dataframe combining the accuracy measurements of all classifiers with the Random/Grid Search approach was created (see **Figure 25**).

Following this approach, the performance of 11 models out of 15 improved. The drop in accuracy of the four models (Histogram Gradient Boosting Classifier, SVM, Logistic Regression, and Naive Bayes) can be explained by the Random Search that missed catching the default values that could also be the optimal parameters. The only way to reach the optimal parameters is to run the Grid Search on all values which is not possible with the available computational power.

Assessment

The table below displays the best performance of each model whether under the Default, Random, or Grid Search parameters.

Model	F1 Score	Precision	Recall	Accuracy
Gradient Boosting Classifier	1.00	1.00	1.00	1.00
Multi-layer Perceptron Classifier	1.00	1.00	1.00	1.00
SGD Classifier	1.00	1.00	1.00	1.00
Histogram Gradient Boosting Classifier	1.00	1.00	1.00	1.00
Random Forest	0.9958	0.9957	0.9957	0.9957
Stacking Classifier	0.9958	0.9957	0.9957	0.9957
Bagging Classifier	0.9958	0.9957	0.9957	0.9957
Extra Trees Classifier	0.9958	0.9957	0.9957	0.9957
KNN	0.9958	0.9957	0.9957	0.9957
Voting Classifier	0.9958	0.9957	0.9957	0.9957
Decision Tree	0.9873	0.9875	0.9871	0.9871
AdaBoost Classifier	0.9536	0.9531	0.9528	0.9528
SVM	0.8797	0.8755	0.8755	0.8755
Logistic Regression	0.8675	0.8602	0.8584	0.8584
Naive Bayes	0.8617	0.8548	0.8541	0.8541

Another indicator: ROC-AUC (Receiving Operating Characteristic - Area under the curve) was used to assess the models performance (see **figures 26** to **38**) .

In addition,In order to test the robustness of the model, Gaussian noise or white noise have been applied. This process is more commonly known as jitter. Many studies have noted that adding small amounts of input noise (jitter) to the training data often aids generalization and fault tolerance. (Russell Reed and Robert J Marks II, 1999) **[1]**.

In this manner, the noise has been applied for the models with better accuracy. For better interpretation, the accuracy for the models have been plotted, reflecting the performance while jitter is increasing. Between 0% and 15% noise MLP Classifier is the best model, from 15% and 40% noise Random Forest is the best model and in between 40% and 50% noise SVM Classifier results to be the best one (see **Figure 39, 40**).

In summary, the model with the best overall performance is the Random Forest Classifier as it is the most stable one through the noise variations . It was decided to use the **Random Forest** with the following parameters: **(bootstrap=False, max_depth=60, max_features='auto', min_samples_split=6, n_estimators=180, random_state=15)**.

Conclusion & Discussion

In this project, it was asked to develop a machine-learning model to predict the likelihood of a patient suffering from SP based on Sociodemographic, Health-related, and Habit-related data.

It was a perfect opportunity to apply theoretical knowledge in a practical case. It was observed also that there are no general rules to apply that work in every situation. The process required some subjective choices, an iterative way of working, and a trial-and-error approach.

In a nutshell, the work was divided into 3 big parts: Data exploration to have a general review of the data, followed by Data Preprocessing to prepare data for the modeling part and fix anomalies, and finally Modeling where several models were tried and several optimization techniques were implemented such as random and grid search to find better parameters for the models.

Finally, it was decided to use **RandomForestClassifier.** Choosing this algorithm with these specific parameters was based on sensitivity analysis to noise (Jitter) in the data.

Some enhancements can be added to the project such as generating pipelines to automate many of the steps involved in building and deploying models, which can save time and reduce the risk of human error and it can make it easier for multiple people to work on the same project . In addition, using more advanced algorithms other than the ones in the scikit-learn library can potentially provide better and more accurate results.

References

Introduction to Machine Learning Third Edition Ethem Alpaydın The MIT Press Cambridge, Massachusetts London, England

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems

Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, 1999. [1]

Input noise injection for supervised machine learning, with applications on genomic and image data [2]

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html [3]

Conceptual Econometrics Using R Jean-Marie Dufour, Julien Neves, in Handbook of Statistics, 2019 [4]

Annexes

1. Creativity & Other Self Study

Noise Injection (Jitter)

Injecting noise in the data is embedded in many ensemble methods as a part of the training process. Adding even more noise when manipulating the examples is shown also to be more e↵ective in terms of performance of the final model . It is indeed obvious to see that adding noise to the data is the main cause of diversity when using the learning algorithm especially if there is no randomness involved during training and in the absence of other randomness such as random initialisations. As shown in the introduction, such diversity can reduce the variance of the ensemble method and improve its generalization performance. However, a mixture of expert methods requires reasonable time for training data for classifiers that is proportional to the number of individual classifiers and their individual computational complexity, hence resulting in high computational cost.

The term noise Injection has been first used in an explicit way in supervised learning in the case of neural networks in the late 80's under the stochastic optimisation scheme which falls under the ERM framework . Other terms such as training with noise, adding noise and jitter were also used . Indeed, inspired by the robustness of biological neural networks to noisy environments and/or the loss of neurons, a large body of work has tried to study and emulate this property for artificial neural networks that was termed as fault tolerance, although also the terms robustness and reliability were equally used. The authors list ways that have been used to introduce uncertainties during training, such as flipping the labels, adding Gaussian noise to the input data or in the synaptic weights. **[2]**

Hyperparameter Tuning

Hyperparameter Tuning is a technique used to tinker with the values of the hyperparameters, in an attempt to find the value combinations that will lead to the best model scores. The tuning methods used in this assignment were Random Search first and then Grid Search using a space of hyperparameters containing the Random Search hyperparameters in that interval. Thus gaining computational power and time of execution(see **code capture 1&2**).

Random search

RandomizedSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter. **[3]**

Grid search

Grid search is a process that searches exhaustively through a manually specified subset of the hyperparameter space of the targeted algorithm. **[4]**

2. Figures

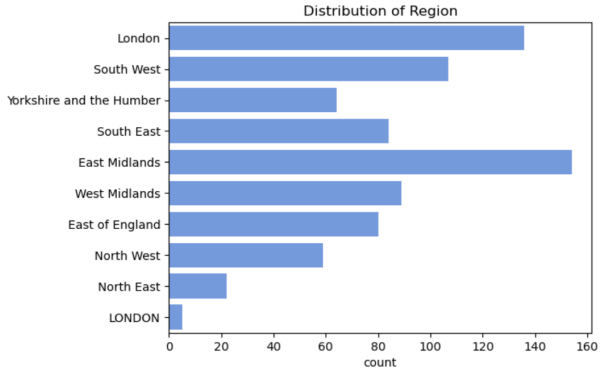

Figure1: Count Plot of 'Region'

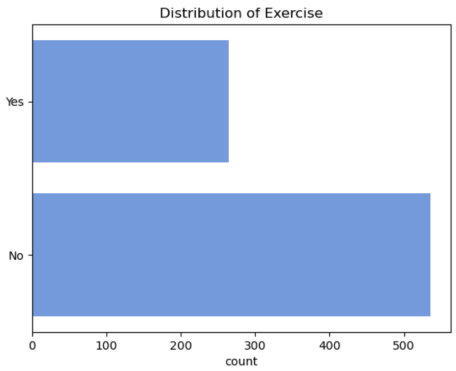

Figure 2: Count Plot of 'Exercice'

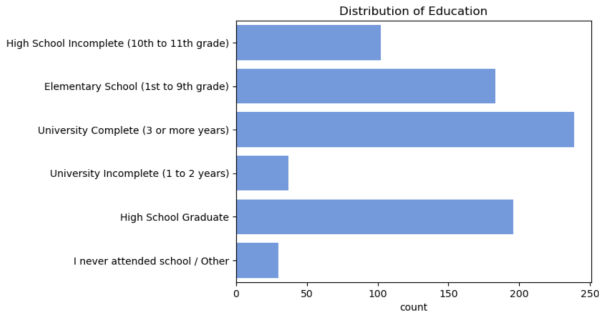

Figure 3: Count Plot of 'Education'

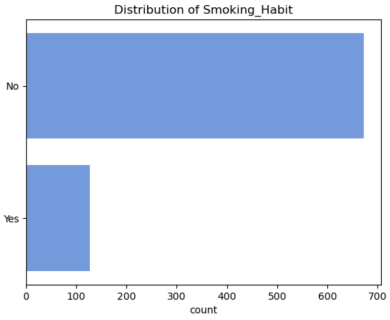

Figure 4: Count Plot of 'Smoking_Habit'

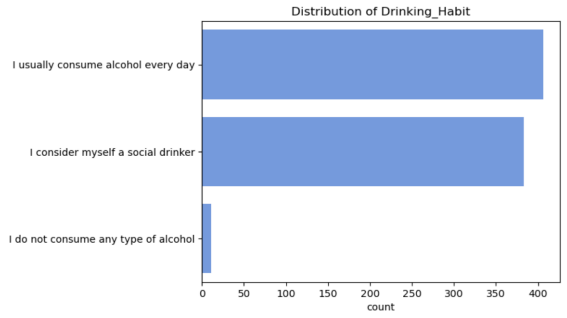

Figure 5: Count Plot of 'Drinking_Habit'

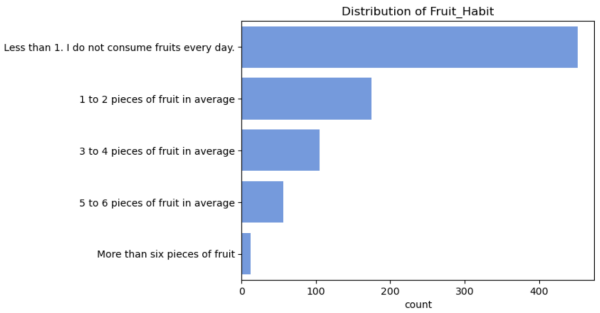

Figure 6: Count Plot of 'Fruit_Habit'

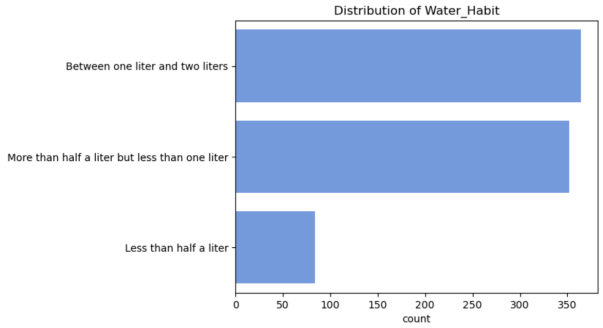

Figure 7: Count Plot of 'Water_Habit'

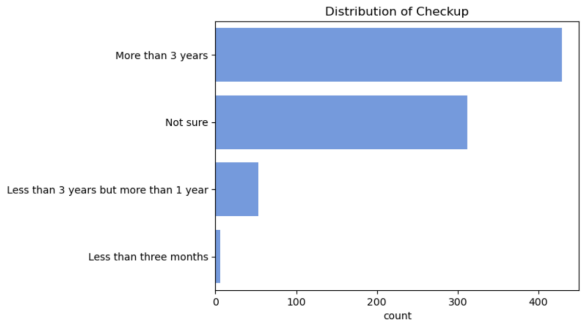

Figure 8: Count Plot of 'Checkup'

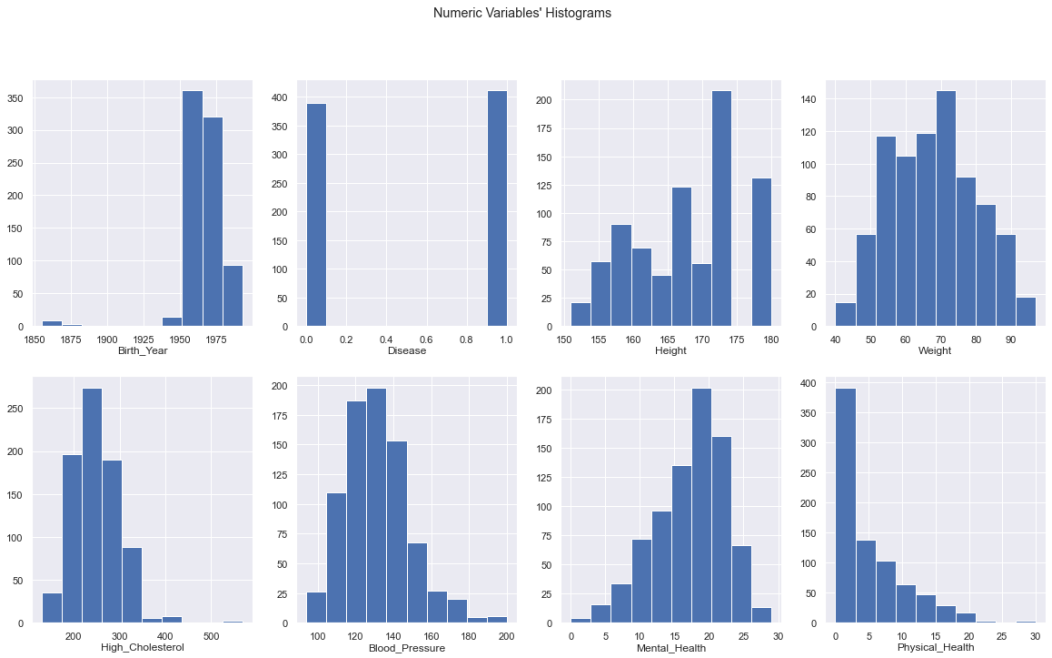

Figure 9: A histogram for each numerical variable

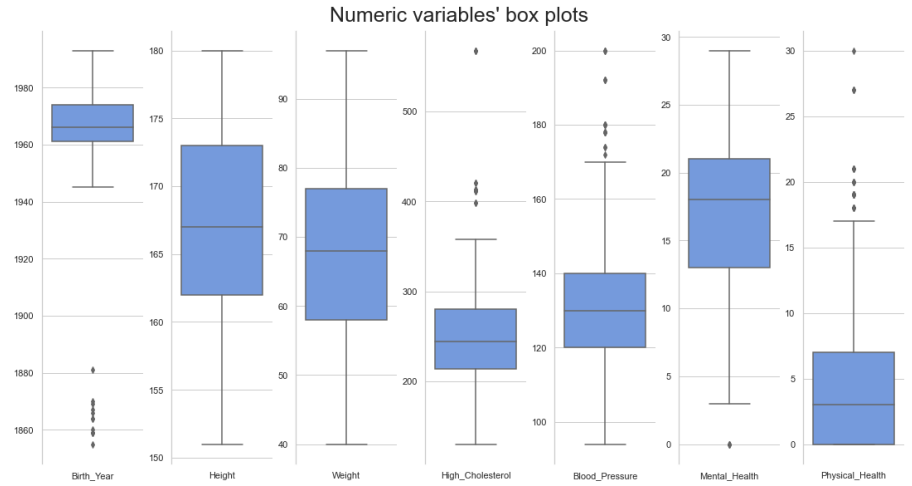

Figure 10: Box plots for numerical variables

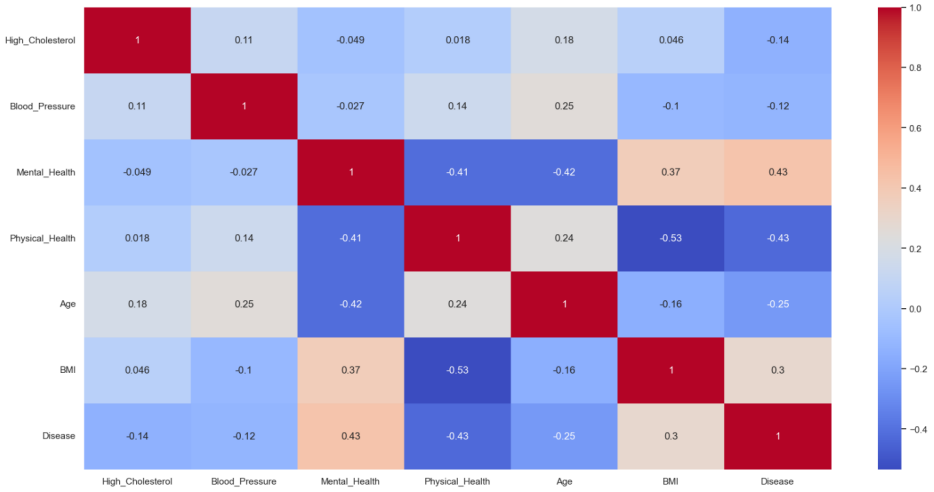

Figure 11: Correlation map of the numeric variable

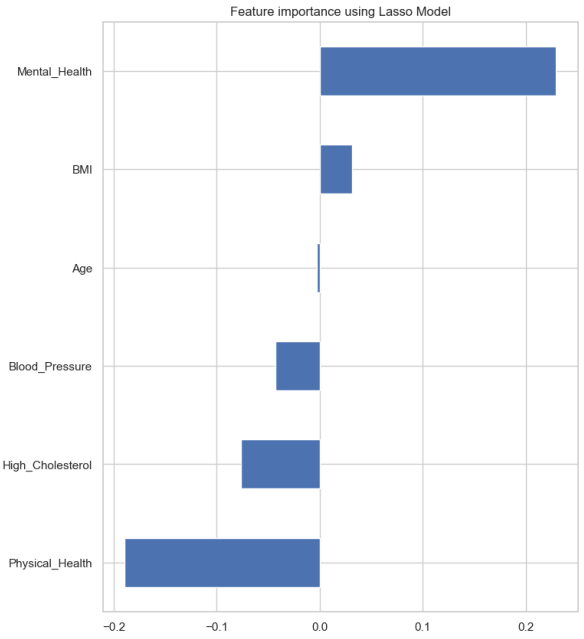

Figure 12: Feature importance using Lasso model

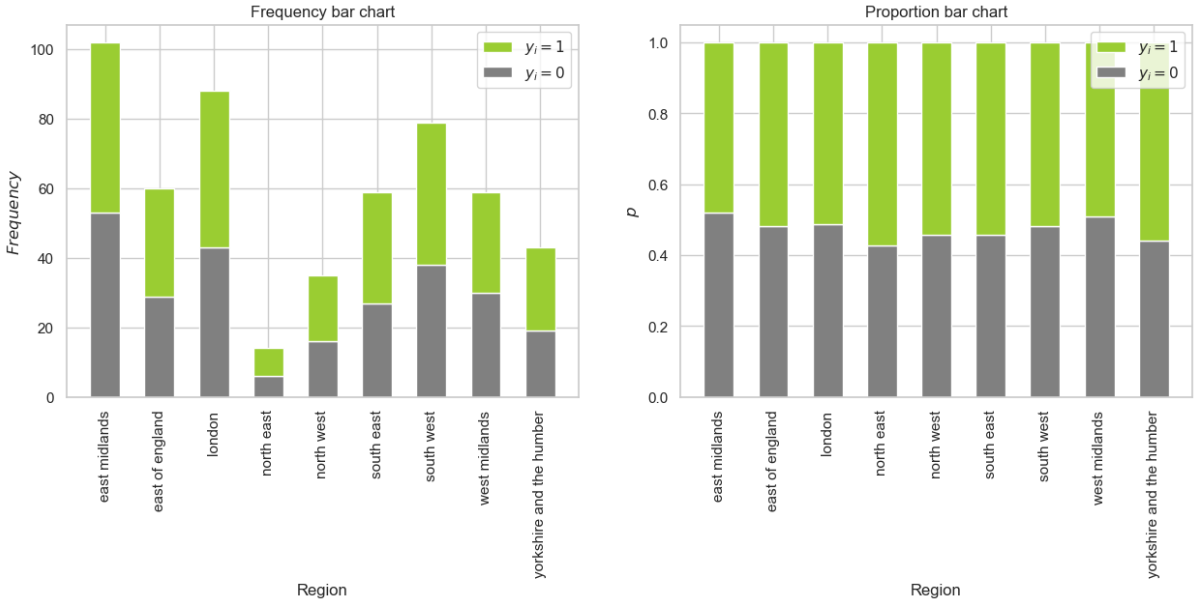

Figure 13: Distribution of target variable across ‘Region’

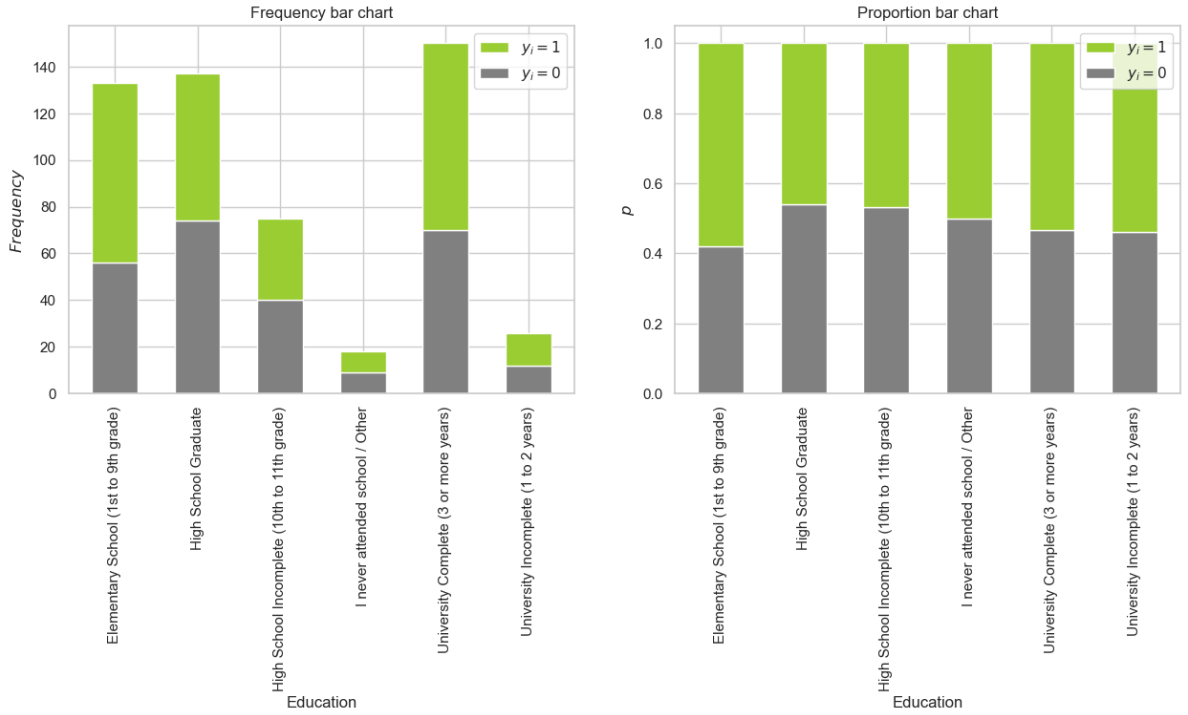

Figure 14: Distribution of target variable across ‘Education’

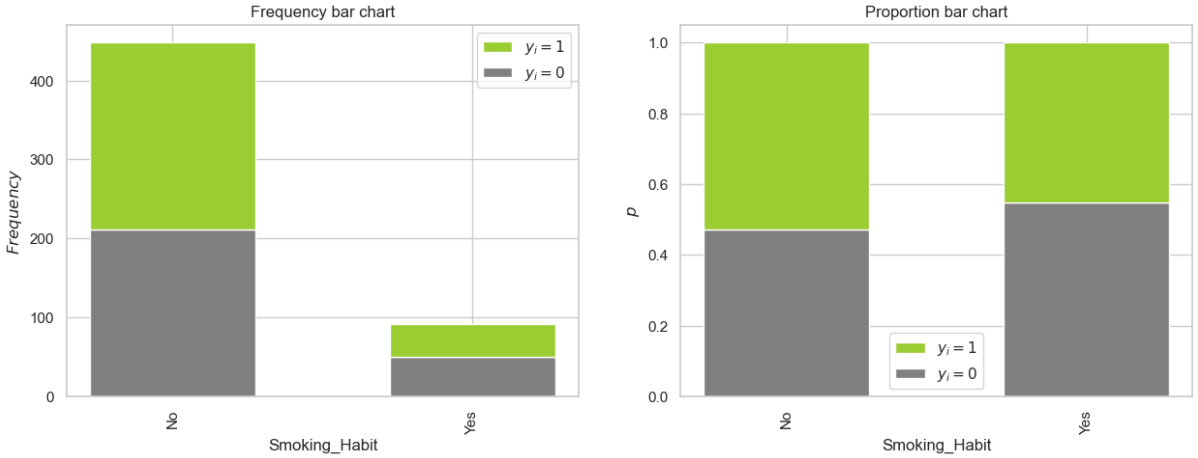

Figure 15: Distribution of target variable across ‘Smoking_Habit’

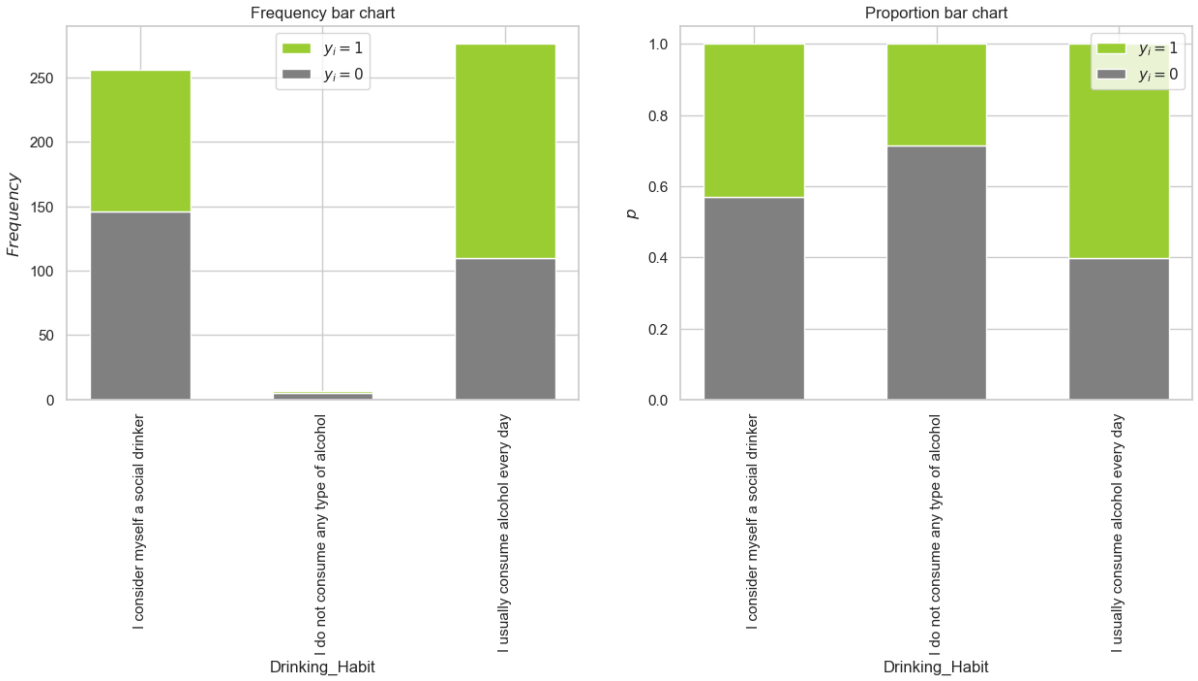

Figure 16: Distribution of target variable across 'Drinking_Habit'

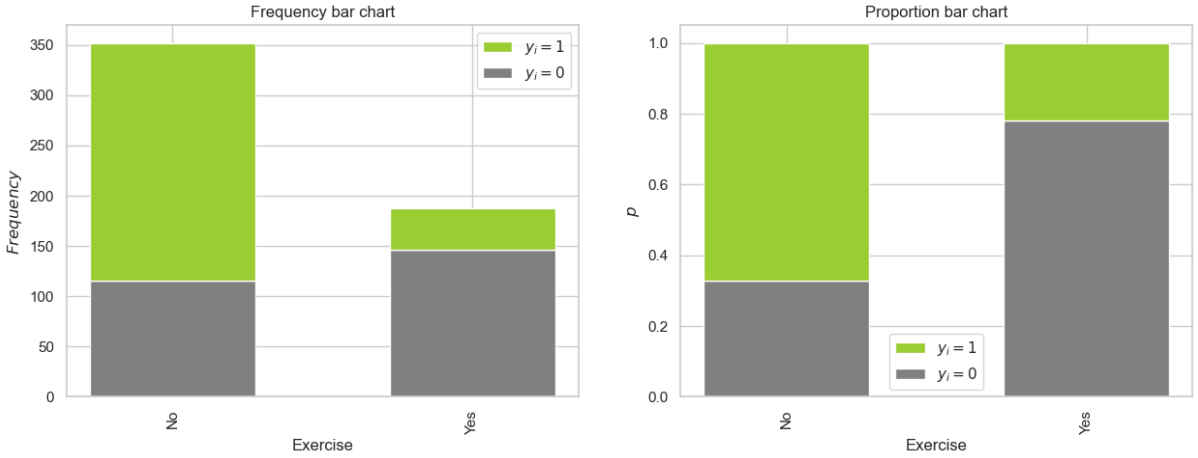

Figure 17: Distribution of target variable across 'Exercise'

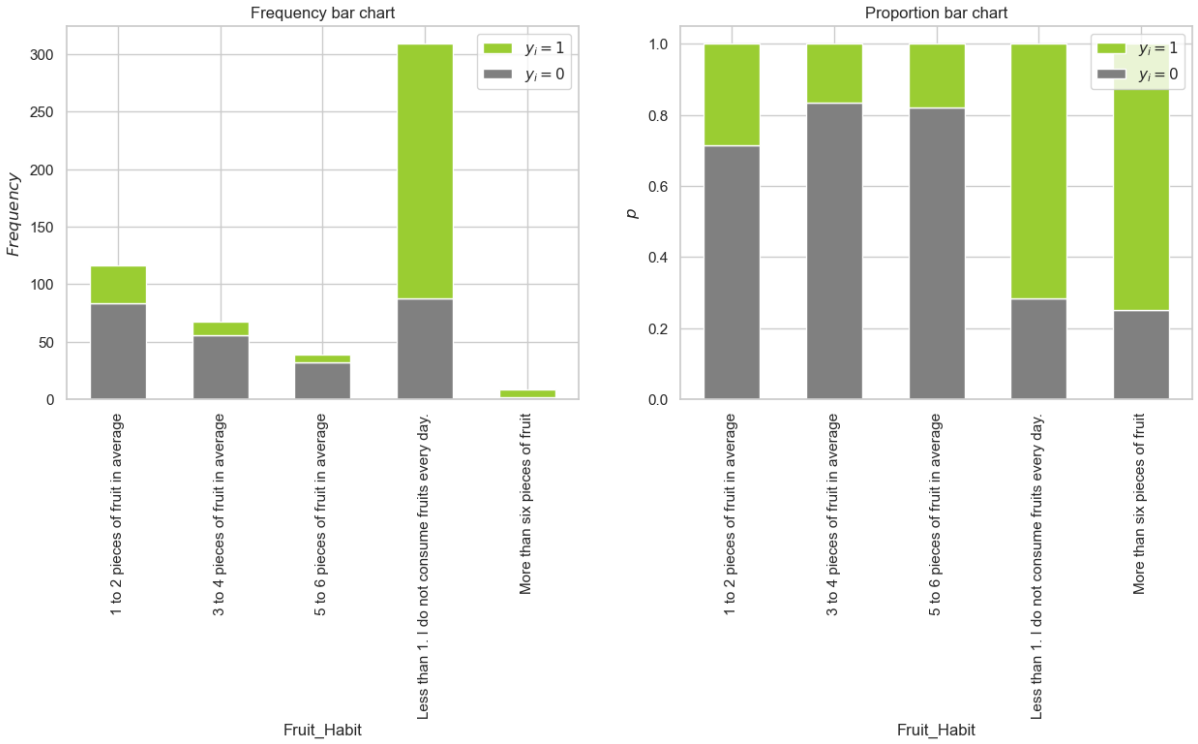

Figure 18: Distribution of target variable across 'Fruit_Habit'

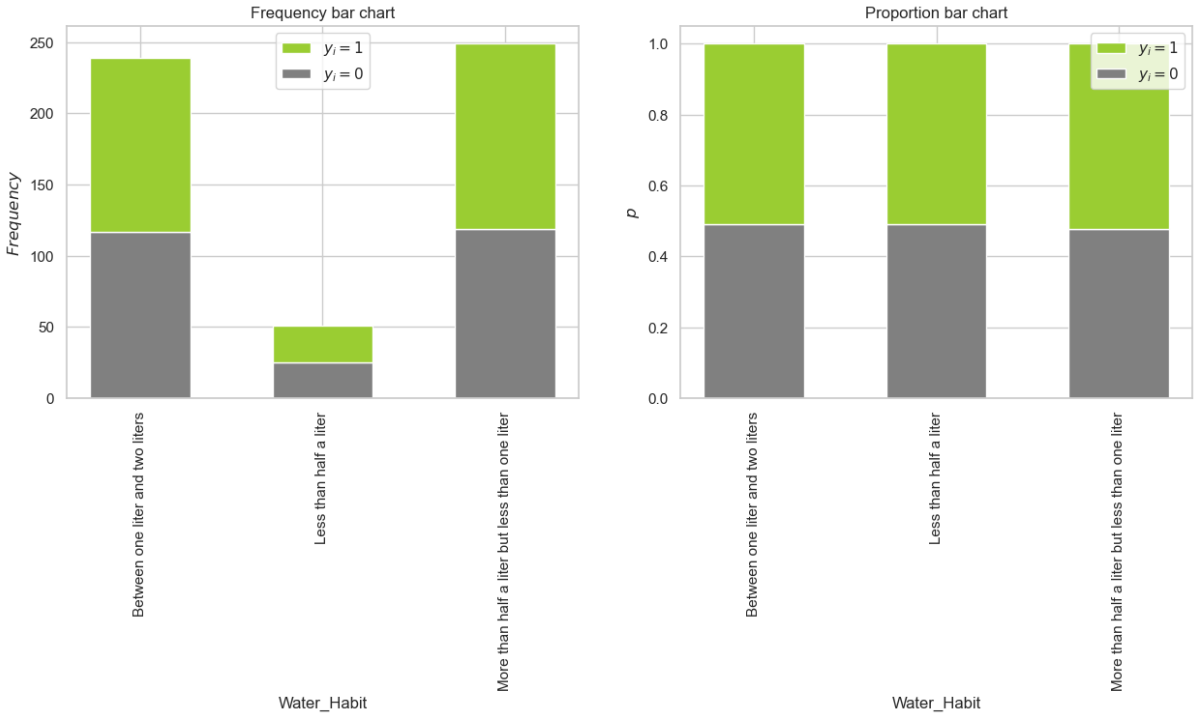

Figure 19: Distribution of target variable across 'Water_Habit'

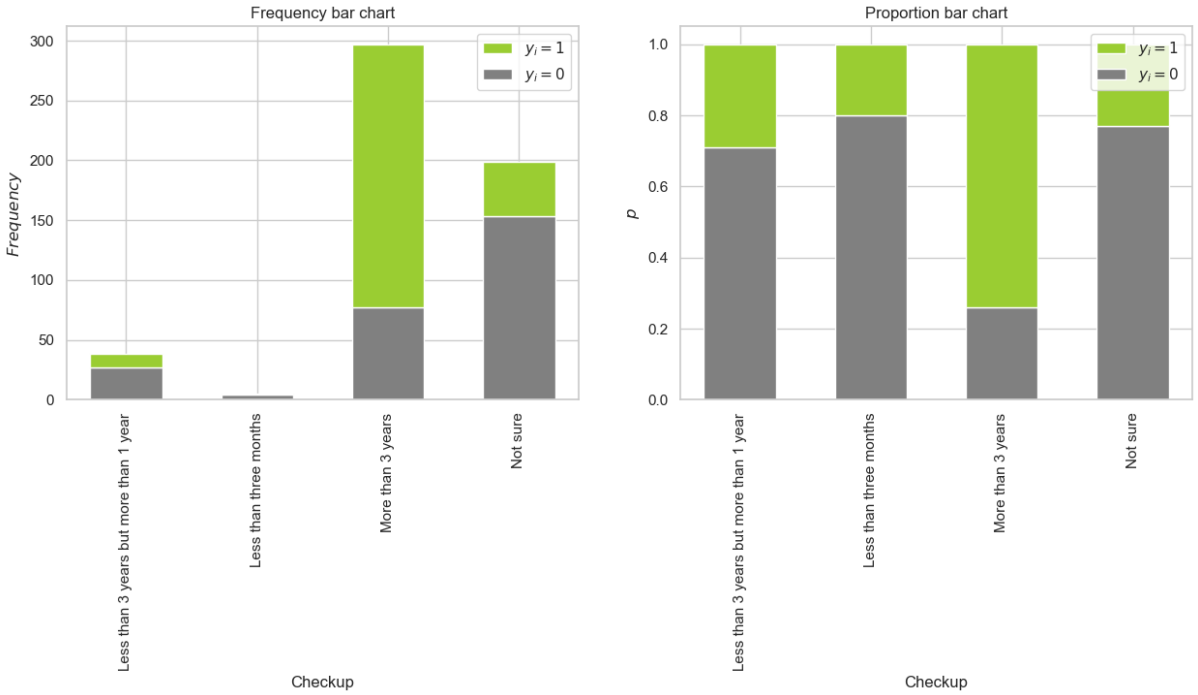

Figure 20: Distribution of target variable across ‘Checkup’

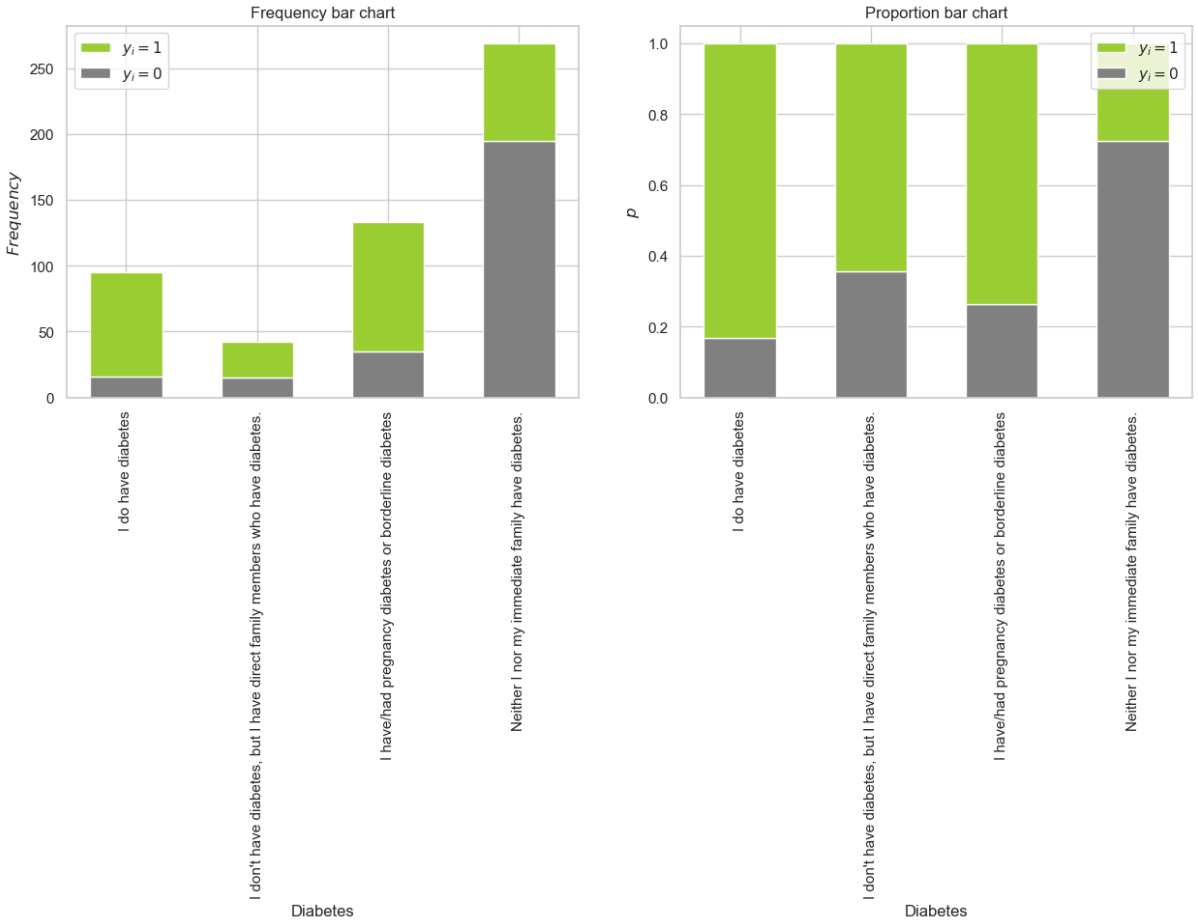

Figure 21: Distribution of target variable across ‘Diabetes’

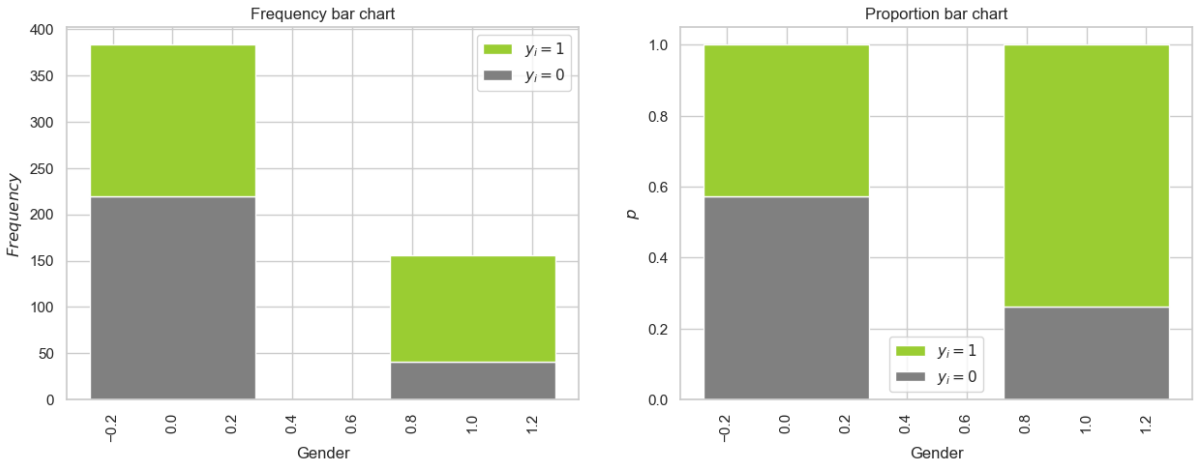

Figure 22: Distribution of target variable across 'Gender'

Model	F1 Score	Precision	Recall	Accuracy
Histogram Gradient Boosting Classifier	1.000000	1.000000	1.000000	1.000000
Stacking Classifier	0.995816	0.995746	0.995708	0.995708
Bagging Classifier	0.991597	0.991566	0.991416	0.991416
Random Forest	0.987342	0.987457	0.987124	0.987124
Extra Trees Classifier	0.987342	0.987457	0.987124	0.987124
Decision Tree	0.983051	0.983420	0.982833	0.982833
Gradient Boosting Classifier	0.957983	0.957238	0.957082	0.957082
Voting Classifier	0.954357	0.952811	0.952790	0.952790
Multi-layer Perceptron classifier	0.920502	0.918507	0.918455	0.918455
AdaBoost Classifier	0.886076	0.884465	0.884120	0.884120
KNN	0.884298	0.879878	0.879828	0.879828
SVM	0.879668	0.875532	0.875536	0.875536
SGD Classifier	0.868526	0.861206	0.858369	0.858369
Logistic Regression	0.867470	0.860200	0.858369	0.858369
Naive Bayes	0.861789	0.854795	0.854077	0.854077

Figure 23: The performance of all models with default parameters

Model	F1 Score	Precision	Recall	Accuracy
Multi-layer Perceptron classifier (Grid Search)	1.000000	1.000000	1.000000	1.000000
Multi-layer Perceptron classifier (Randomized Search)	0.995816	0.995746	0.995708	0.995708
Multi-layer Perceptron classifier (Default values)	0.920502	0.918507	0.918455	0.918455

Figure 24: Example of the performance of one model under the 3 approaches

Model	F1 Score	Precision	Recall	Accuracy
Gradient Boosting Classifier	1.000000	1.000000	1.000000	1.000000
Multi-layer Perceptron classifier	1.000000	1.000000	1.000000	1.000000
SGD Classifier	1.000000	1.000000	1.000000	1.000000
Random Forest	0.995816	0.995746	0.995708	0.995708
Stacking Classifier	0.995816	0.995746	0.995708	0.995708
Bagging Classifier	0.995816	0.995746	0.995708	0.995708
Extra Trees Classifier	0.995816	0.995746	0.995708	0.995708
KNN	0.995816	0.995746	0.995708	0.995708
Voting Classifier	0.995816	0.995746	0.995708	0.995708
Decision Tree	0.987342	0.987457	0.987124	0.987124
Histogram Gradient Boosting Classifier	0.983051	0.983420	0.982833	0.982833
AdaBoost Classifier	0.953586	0.953126	0.952790	0.952790
Naive Bayes	0.858268	0.850027	0.845494	0.845494
SVM	0.857143	0.850238	0.849785	0.849785
Logistic Regression	0.844106	0.836253	0.824034	0.824034

Figure 25: The performance of all models with Random/Grid Search parameters

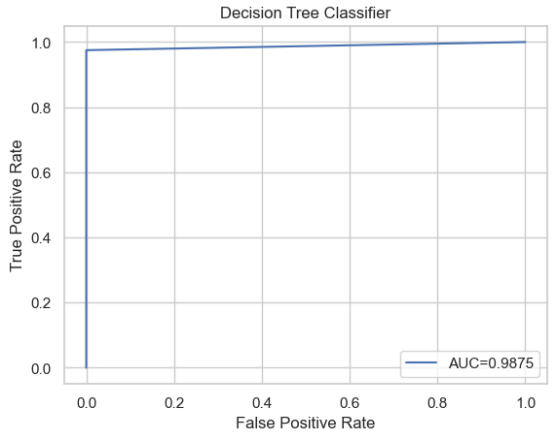

Figure 26: ROC-AUC of DTC

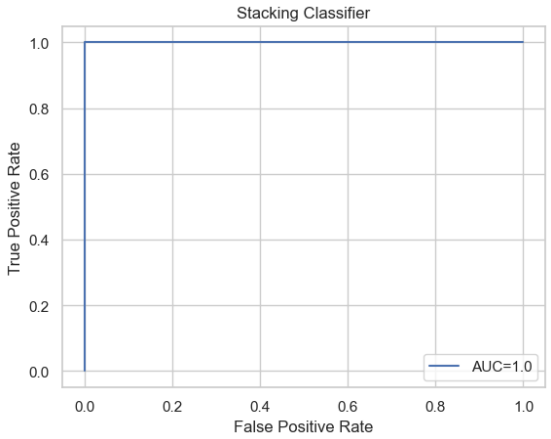

Figure 27: ROC-AUC of SC

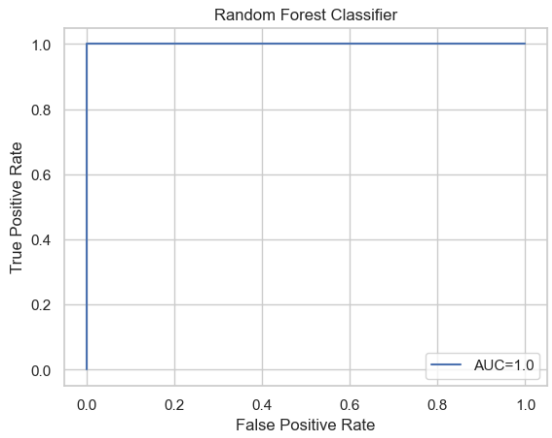

Figure 28: ROC-AUC of RFC

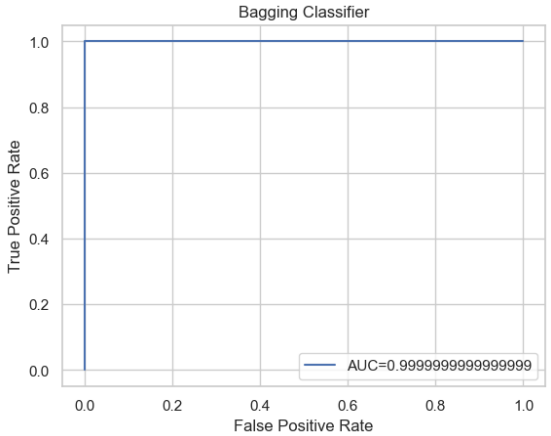

Figure 29: ROC-AUC of BC

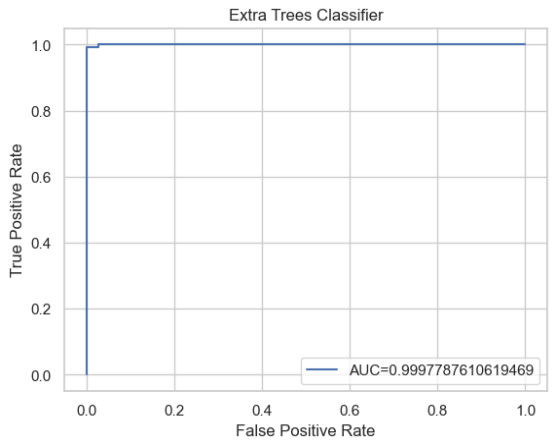

Figure 30: ROC-AUC of ETC

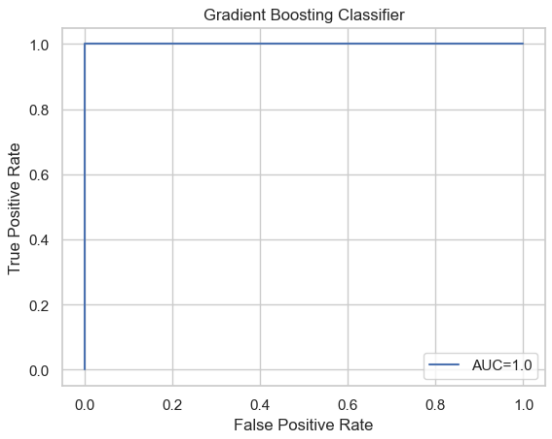

Figure 31: ROC-AUC of GBC

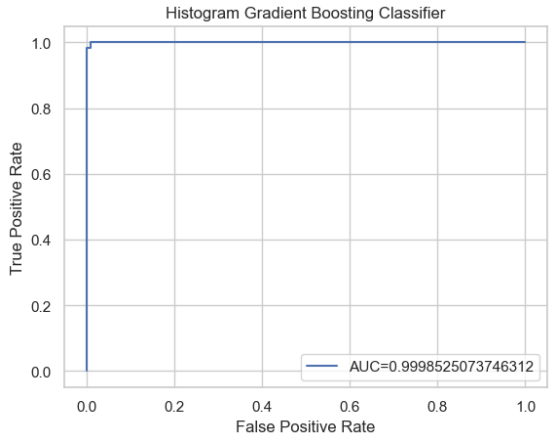

Figure 32: ROC-AUC of HGBC

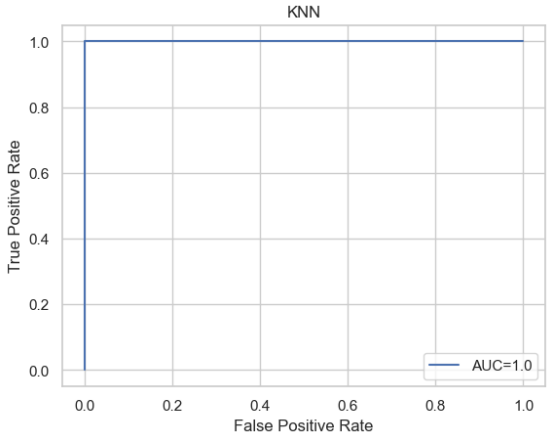

Figure 33: ROC-AUC of KNN

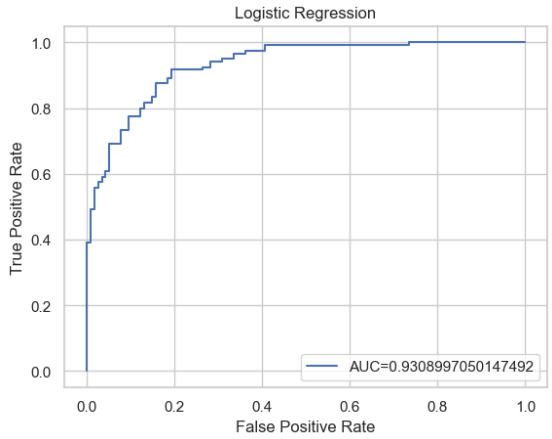

Figure 34: ROC-AUC of LR

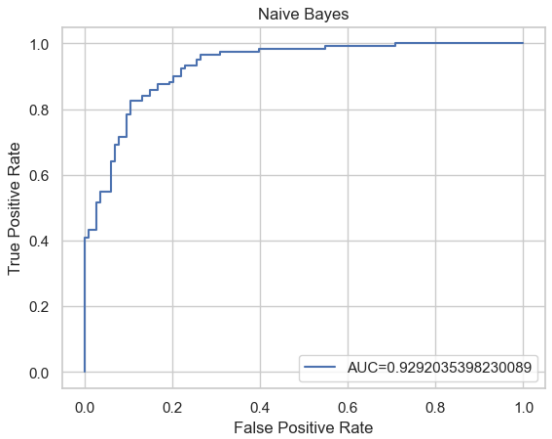

Figure 35: ROC-AUC of NB

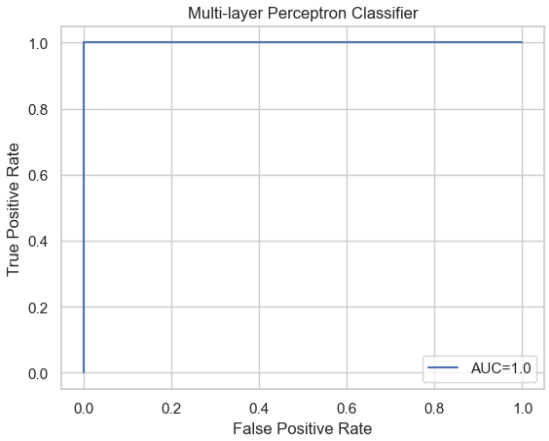

Figure 36: ROC-AUC of MLPC

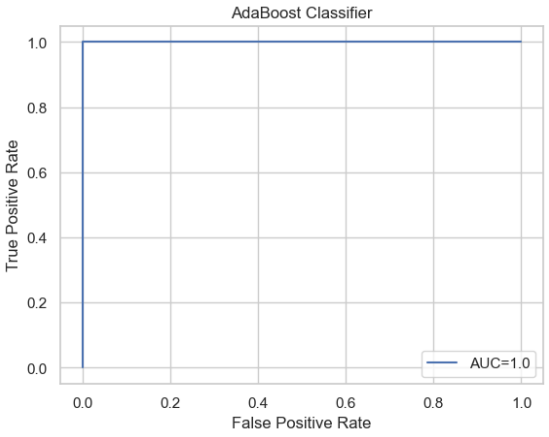

Figure 37: ROC-AUC of ABC

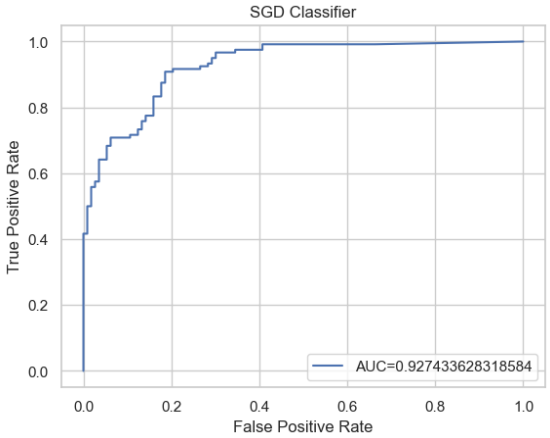

Figure 38: ROC-AUC of SGD

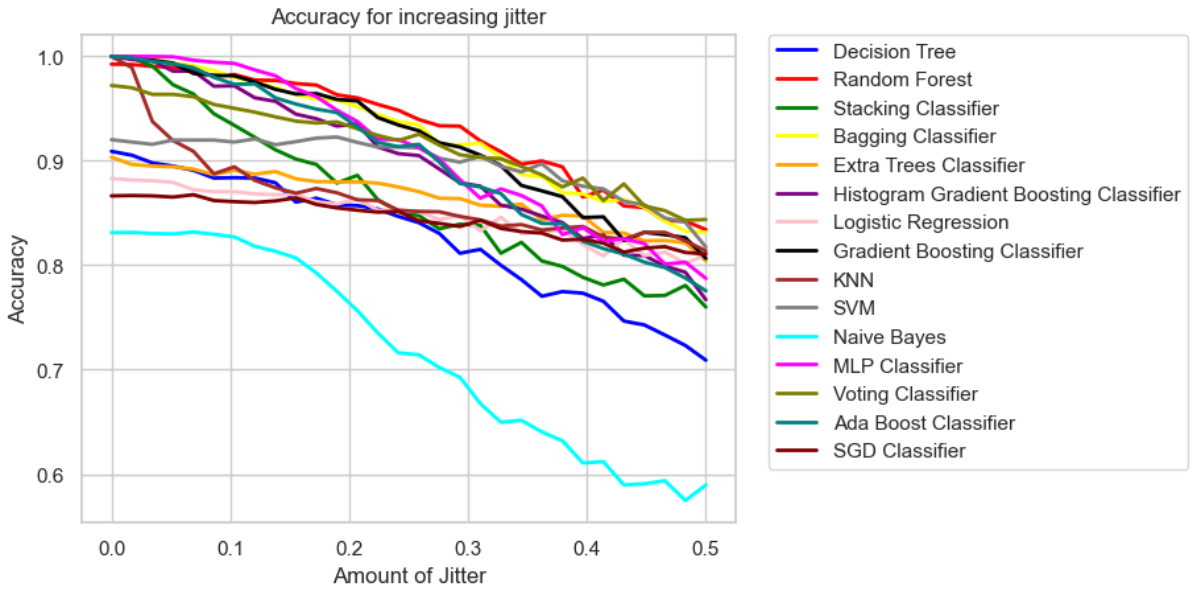

Figure 39: Models' performance applying Jitter

	Score
Random Forest Classifier	0.933210
Bagging Classifier	0.915399
Gradient Boosting Classifier	0.913173
Voting Classifier	0.905751
Support Vector Machine Classifier	0.898701
Multi-layer Perceptron Classifier	0.882004
Histogram Gradient Boosting Classifier	0.879035
Ada Boost Classifier	0.878293
Extra Trees Classifier	0.863822
Logistic Regression	0.847495
KNN Classifier	0.846753
Stacking Classifier	0.839332
SGD Classifier	0.837106
Decision Tree Classifier	0.811503
Naive Bayes	0.692764

Figure 40: Models' performance at jitter noise =0.3

```
# Find best parameters using Randomized search
print('Best parameters for Random Forest Classifier using Randomized search')
model_grid_search_rf_rs = tuning_random_search(RandomForestClassifier(), param_grid_rf, 'Random forest Classifier')
```

```
Best parameters for Random Forest Classifier using Randomized search
Fitting 5 folds for each of 500 candidates, totalling 2500 fits
|                   | Assigned Value   |
|:------------------|:-----------------|
| bootstrap         | False            |
| max_depth         | 10               |
| max_features      | auto             |
| min_samples_leaf  | 2                |
| min_samples_split | 6                |
| n_estimators      | 600              |
| random_state      | 15               |
```

Code Capture 1: Example of Random Search on RandomForestClassifier

```
# Using the best parameters found in the previous step, create a grid for Grid search
param_grid_rf_rs = {'bootstrap': [True, False],
                'max_features': ['auto', 'sqrt', 'log2'],
                'n_estimators': [180,190,200,210],
                'max_depth': [60,65,71,75],
                'min_samples_split': [5,6,7,8],
                'min_samples_leaf': [1,2,3],
                'random_state': [15]}
```

```
# Find best parameters using Grid search
print('Best parameters for Random Forest Classifier using Grid search')
model_grid_search_rf_gs = tuning_grid_search(RandomForestClassifier(), param_grid_rf_rs, 'Random forest Classifier')
```

```
Best parameters for Random Forest Classifier using Grid search
Fitting 5 folds for each of 1152 candidates, totalling 5760 fits
|                   | Assigned Value   |
|:------------------|:-----------------|
| bootstrap         | False            |
| max_depth         | 60               |
| max_features      | auto             |
| min_samples_leaf  | 1                |
| min_samples_split | 6                |
| n_estimators      | 180              |
| random_state      | 15               |
```

Code Capture 2: Example of Grid Search on RandomForestClassifier