The Smith Parasite's Predictive Model

Afonso Reyna (r20191197), André Silva (r20191226), Dinis Melo (r20191190), Gonçalo Rodrigues (r20191300), Mafalda Martins (r20191220)

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS 22/23

MACHINE LEARNING - NOVA INFORMATION MANAGEMENT SCHOOL

ABSTRACT: Dr. Smith in England has just now identified a brand-new illness. We have been asked to investigate this. More than 5000 people have already been afflicted by the illness, with no obvious connection between them. Fever and fatigue are the most typical signs, but some infected individuals show no symptoms at all. In any case, this virus is linked to post-disease symptoms including speech loss, confusion, chest pain, and shortness of breath. It is yet unclear how the sickness spreads, and there is no way to predict whether a patient would develop the illness or not. However, certain populations appear to be more likely than others to contract the parasite. The data was treated and tested in different ways – missing values, duplicates, outliers and incoherencies were handled, and features were selected, transformed and created. We've finally reached a final model with *Random Forest Classifier* and achieved an extremely high degree of prediction ability, getting an F1 score of 1.

1. Introduction

Smith Parasite is a disease that has infected 5000 people, first discovered by Dr. Smith. Many of the patients felt continuous fatigue and fever, others were simply asymptomatic. The most worrisome fact is the sequelae that have been presenting, including loss of speech, confusion, chest pains, and shortness of breath. To this day, factors that are favorable to contracting this disease are still unknown, although it seems that certain groups of people have an easier time becoming infected.

Building a predictive model to identify traits of individuals who are more prone to get the disease is the aim of this study. The two datasets provided to us - training and test - are composed of records of people and their characteristics, including whether or not they are infected with the disease. These characteristics include variables such as physical characteristics, mental health, smoking or not, physical exercise, among others, which will allow us to understand which population groups are more likely to contract this disease. The target variable is 'Disease', and it can be 0 - if the person is not infected with the virus -, or 1 - if the person is infected with the virus.

2. Exploration

After obtaining the data, we started working on analysing the dataset in order to better understand its characteristics and relationships between variables. The very first thing we did was to define the variable 'PatientID' as the index of our dataset. In the Annex, we can check the initial variables that composed our dataset (*Table 1*). We began to look at its shape and data types of the variables to get an idea of what we were dealing with. Also, we calculated some simple statistics such as the median, mean, and standard deviation to get a general sense of the data.

With this analysis, we noticed that we had 13 missing values in only one variable, ‘Education’, which will be treated later in the data pre-processing. We also checked the presence of duplicated rows, that could cause overfitting in our model. Fortunately, our dataset did not have any. At this point, our data has 800 observations and is composed of 18 variables, 8 of them being numerical and 10 categorical, whereby one of the numerical variables is our target - ‘Disease’.

Since it is very important in this initial phase to have a solid understanding of the data, we performed some basic visualizations, such as histograms, and noticed from the beginning there were some changes that could be made, namely the creation of some variables. So, we believed that it was beneficial to firstly apply these changes, and then perform new and more detailed visualizations with the complete dataset.

3. Pre-processing

Moving on to the data pre-processing, where all the steps that are needed to transform the raw data into the data prepared to model are included. These steps go from the creation of new variables to the cleaning and reduction of our data.

Therefore, in order to make the data more meaningful for our project, we created several new variables. We started by creating a gender (with the name ‘male’) variable based on whether a person's name started with "Mr." or "Mrs.". This variable gets the value 1 if the person is a male and 0 if female. Then, we dropped the variable ‘Name’ because it became useless. By subtracting a person's birth year from the current year (2022), we created the variable ‘age’, since we believed it was more useful to work with people’s age instead of their birth year. Consequently, we dropped ‘Birth_Year’. We also created a ‘IMC’ (Body Mass Index) variable using a person's weight and height. From this new variable, we also created one that is called ‘is_fat’, a binary variable that gets the value of 1 if the ‘IMC’ is above 25 or 0 otherwise. Finally, using the categorical variable ‘Diabetes’, we created two binary variables, ‘Diabetes_History’ and ‘Diabetes_Family’ to simplify the analysis. The first one gets the value 1 if the person has diabetes or if he/she has or had pregnancy or borderline diabetes, and 0 otherwise. The second variable concerning family distinguishes the people that have direct family with diabetes (1) and the ones that don’t (0).

Next, we took a look at the data to see if there were any inconsistencies. We found that the variable ‘Region’ contained "London" and "LONDON" as different entries, so we decided to lowercase everything to avoid any confusion, since both have the same meaning.

Once we had defined our new variables, that we can check in the Annex (*Table 2*), we proceeded to perform some visualizations *(Fig. 1 to 10).* We started to visualize the relationships between the different variables and our target *variable, Disease*. For that, we used histograms to see how the target means were distributed in the different variables in order to identify any potential insights (*Fig. 1 to 4*). From our visualizations, we were able to see that certain variables had a stronger impact on the likelihood of developing the disease. For instance, we found that people with higher ‘IMC’ were more likely to develop the disease – which led

us to the creation of ‘is_fat’. Finally, we checked the pairwise relationships between the variables to see if there were any interesting remarks (*Fig. 5*). We found that ‘Weight’ and ‘Height’ were highly correlated, for example, but we decided to keep them both for now.

In order to understand the presence of outliers in our data, we checked the Histograms and Boxplots for all our numeric variables (*Fig. 6 and 7*), where, at first sight, we checked that 5 variables contained possible outliers, being them ‘High_Cholesterol’, ‘Blood_Pressure’, ‘Mental_Health’, ‘Physical_Health’ and ‘age’. Once we took a closer look, we noticed that the observations primarily considered as outliers in both ‘Mental_Health’ and ‘Physical_Health’ are plausible values, since these variables can have values that go from 0 to 30, whereby any of those values fall out of this interval. Regarding ‘High_Cholesterol’, there were 3 observations that had the value of 568 that caught our attention, since it is a value that seems farfetched and very isolated from the distribution, and suspicious that 3 different people had the exact same elevated number. As for the remaining outliers for ‘High_Cholesterol’ and ‘Blood_Pressure’, we believe that, despite these values are too high to an unhealthy point, they are realistic, and their presence could make a difference in the prediction. In what concerns the variable ‘age’, there were 12 observations with values higher than 120, which we considered to be really outliers because human life does not last that long.

Since we do not have many observations to work with, instead of deleting the outliers, we decided to perform the *KNN Imputer* method on these values. This method predicts the values for the missing data in a variable by considering the values of the other similar rows of the dataset and comparing them with the non-missing values in that same variable.

Therefore, as we are dealing with extreme values and not non-existing ones, we started by turning these 12 observations from ‘age’ and 3 from ‘High_Cholesterol’ that we considered as outliers into NaNs.

As mentioned before, there is only one feature that contains missing values before our actions in treating outliers, and that is ‘Education’. For the same reason as before, we decided to apply the KNN Imputer method (*details in the annex)* on these 13 observations, instead of just deleting them.

After treating the outliers and missing values, we proceeded to the aggregation of categories in the categorical variables, which involves reducing the number of some variables’ classes, by aggregating them, in order to have simpler variables and easier to be dealt with. This is an important step to make it easier to later pass the categorical variables to numerical ones, in particular when creating dummy variables, because this process will add to the dataset the number of categories the variables have minus one. Thus, we only applied this type of change in 3 of the 8 categorical variables, being them ‘Checkup’, ‘Water_Habit’ and ‘Drinking_Habit’. In the case of variables with only 2 categories, there is no need or even sense to aggregate them, and when passed to dummy variables, no further variable is created. In what concerns ‘Region’, ‘Education’ and ‘Fruit_Habit’, we did not consider the aggregation of categories, since their classes were more or less reasonably distributed among the 800 observations.

In the end, for ‘Checkup’, we decided to only have 3 categories (initially had 4), being them “More than 3 years”, “Not sure” and “Less than 3 years”. This last category was created by grouping the other 2 categories that were removed, one of them only contained 6 observations, so we believed that this change would not have a big impact. Regarding ‘Water_Habit’, we considered it only mattered to distinguish if a person drinks more or less than a liter of water a day, especially because only 84 people (in 800) drink less than half a litre of water per day. Finally, in ‘Drinking_Habit’, we decided to have only 2 categories, being them “daily” and “I do not consume alcohol daily”. From the initial categories, the observations that had “no”, which were only 11, as well as the ones that consume alcohol in social gatherings, were grouped together.

Having the variables’ categories reduced, we moved to turning the categorical variables to numerical. In what concerns the variables with initially only 2 categories and the variables to which we performed category aggregation, we created dummy variables.

We used three different methods to convert categorical variables to numerical variables: *Frequency Encoding*, *Target Mean Encoding*, and *Dummy V*ariable Creation. *Frequency Encoding* involves replacing categorical values with the frequency of their occurrence in the dataset. It can be useful for adding information about the importance of each category to the model, but it can also create a bias if the frequencies are not representative of the underlying data distribution, giving more importance to categories that have more frequency. *Target Mean Encoding* involves replacing the categorical values with the mean value of the target variable for that category. *Dummy variable creation* is a way of encoding categorical variables for use in machine learning models. It is important because many algorithms cannot handle categorical variables directly and require them to be transformed into a numerical form. The best results were obtained when we converted all categorical variables to dummy variables.

3.1. Data Partition

Throughout the project, we employed different approaches for data partitioning. Initially, we used the *hold-out method*, dividing the data into a 70% train set and a 30% validation set for feature selection. We chose this approach due to the relatively small size of the dataset and the potential complexity of the future analysis and computational demands if we used a cross validation approach for feature selection. However, for model performance evaluation, we employed *K-Fold Cross Validation* with a value of K=10 which used all the training dataset. We believe that this method would provide more robust results and require less analysis effort compared to the feature selection process since in this case we were dealing with just numbers (scores) and not a collection of features.

3.2 Data Standardization

For standardization, we created a function that applied to a specified dataset a specified standardization method. We had 3 options for standardization: *MinMax Scaler*, *Robust Scaler* and *Standard Scaler*. We

thoroughly tested all three options. Theoretically, *Standard Scaler* seemed like the best fit for our decisions, due to the fact that we opted to keep some observations that fell outside the interquartile range (x1.5) - values that could be considered outliers. It allows us to de-emphasize the impact of these values in the data. Ultimately, the latter method managed to yield the best results, and therefore we proceeded with this scaling for the final model.

3.3 Feature Selection

In the Feature Selection phase, we applied several different methods to identify feature importance for categorical and numerical variables alike. Starting with Filter Methods, we checked if there were any univariate variables, that is, with variance equal to zero. Then we proceeded to look at the correlations between variables, using both the *Spearman* and *Pearson* Correlations.

As for categorical variables, we performed the *Chi-Square*, which tells us which of the categorical variables are important to predict the target variable and which are not. We also did the *ANOVA* test, returning the features with the best *ANOVA* F-values, which indicates these should be kept and the remaining features removed. Finally, we applied the *Mutual Information Coefficient (MIC)*, which consists in the measure of how much knowledge a variable can gain from another one. Therefore, variables with a higher value for the *MIC* coefficient should be kept because they have a bigger impact on the target. Moving to Wrapper Methods, we first took into consideration the *Recursive Feature Elimination (RFE)*, using as estimator a Logistic Regression, which returns the optimal number of features and the respective score. Then, we also performed *Forward, Backward and Stepwise Selection*, in which features are added and/or deleted in order to get the best score. Regarding Embedded Methods, we used *Lasso* and *Ridge Regression*. In both of them, variables with the highest coefficients (module value) are considered the most important and the ones that have a coefficient equal to or close to zero should be removed. We also applied *Decision Tree for Feature Importance*, that attributes a score to each variable based on its importance. Here, we compared the *Gini* Coefficient and *Entropy*, in order to formulate better conclusions, where we observed that the variables presented similar values for both the coefficients and in which higher values represent a greater importance. Finally, we applied one method of dimensionality reduction, the *Principal Component Analysis.* An explanation about each of these feature selection methods is provided in the Annex, as well as the features selected by each method (*Table 3*).

Upon thorough analysis of the various techniques implemented and taking a last look at the correlation charts, it became evident that the variables of ‘Weight’, ‘Height’, and ‘IMC’ Index were highly correlated and therefore we should only keep one of them. This is not surprising, as ‘IMC’ is derived from ‘Weight’ and ‘Height’, thus containing some overlapping information. Additionally, it is expected that an individual's weight will increase as their height increases. Given these considerations, we sought to determine which of these variables possessed the greatest predictive power. Ultimately, it was determined that ‘IMC’ was the most robust and consistently selected by the feature selection models, leading us to the decision to

exclude ‘Weight’ and ‘Height’ from the final analysis. These were the only numerical variables that we removed from the dataset, as the other ones didn’t present any problems and proved to be statistically significant.

It was necessary to also selectively eliminate some of the 27 dummy variables generated from the encoding of categorical variables in order to improve the model's performance. To do this, we utilized the previously mentioned techniques. These methods allowed us to assess the strength of the relationship between each dummy variable and the target variable, as well as the overall contribution of each dummy to the model. Lastly, this process resulted in a reduction of the number of dummy variables from 27 to 9, effectively streamlining the model and improving its predictive capabilities.

Below, the final array of features selected can be seen:

Final Variables of Feature Selection			
High_Cholesterol	age	Exercise_Yes	Fruit_Habit_3 to 4 pieces of fruit in average
Blood_Pressure	IMC	Checkup_more_than_3_years	Fruit_Habit_5 to 6 pieces of fruit in average
Mental_Health	gender	Checkup_not_sure	Fruit_Habit_Less than 1. I do not consume fruits every day.
Physical_Health	Diabetes_History	Drinking_Habit_daily	

In order to properly evaluate the performance of our model, we applied the same pre-processing techniques to the test dataset that we used on the training dataset. This included creating the new features, encoding categorical variables, scaling numerical variables and more. However, we made an exception when it came to outliers. While we may have removed or transformed outliers in the training dataset to improve model performance, it would not make sense to do the same in the test dataset. This is because the test dataset represents unseen data, and we want to simulate the real-world conditions as closely as possible. Therefore, we kept the outliers in the test dataset and let the model make predictions on them as-is. The test dataset also did not have any missing values to handle.

4. Modelling

As we progress towards the goal of accurately predicting disease outcomes for patients, it was then necessary to apply the knowledge and techniques learned thus far to models that could effectively achieve this objective. The initial step in this phase was to select the most suitable models for this purpose. After some consideration, our team determined that the following set of models would be considered: *Decision Tree, Logistic Regression, Neural Networks, Gaussian Naive Bayes, Gradient Boosting, Random Forest, Support Vector Machine, Adaptive Boosting* with *Random Forest* as based estimator, *Bagging with Random Forest* as based estimator, *Support Vector Machine*, Stacking with *Neural Networks*, *Random Forest* and *Support Vector Machine* as estimators, *Linear Discriminant Analysis*, *Quadratic Discriminant Analysis*, *K-Nearest Centroid*, *Passive-Aggressive Classifier*, and finally *Voting Classifier*.

In an effort to thoroughly explore new potential solutions, other than the models discussed in class, we also searched for models that were not discussed in classes, including *Linear Discriminant Analysis, Quadratic*

Discriminant Analysis, K-Nearest Centroid, Passive-Aggressive Classifier, and *Voting Classifier*. Detailed descriptions of these models can be found in the Annex. It is important to reinforce that, as previously mentioned in the Data Partition section, a *K-Fold Cross Validation* (with a 10 folds) was used to evaluate the models during this phase. This method allowed us to conduct an accurate assessment of model performance by leveraging the entire dataset rather than relying on a single train-test split.

Subsequently, we conducted evaluations of each model's performance on our dataset without specifying any hyperparameters in order to gauge the inherent capabilities of each model. The results indicated that *Bagging Classifier, Gradient Boosting, Voting Classifier, Stacking Classifier, Decision Tree* and *Random Forest* were the most effective models (Fig. 16).

With an initial understanding of which models exhibited the highest level of performance, we sought to further improve their accuracy through the selection of appropriate hyperparameters. However, rather than solely focusing on the top-performing models, we also wanted to determine if alternative models could surpass the current leaders through the manipulation of their hyperparameters.

To efficiently identify the optimal hyperparameter configurations, we employed a two-pronged approach. First, we used *Random Search* to broadly identify promising hyperparameter combinations, and then employed a *Grid Search* to more precisely identify the most suitable options for each model. This approach allowed us to thoroughly explore the hyperparameter space while minimizing the time and resources required for the optimization process.

The *Random Search* algorithm was implemented with a search budget of 100 iterations, utilizing cross-validation with 3 validation sets per iteration. This resulted in a total of 300 iterations being performed per model. The output of this process, (Table 4) was then used to create another set of hyperparameters but much more reduced to guide the subsequent grid search.

Grid Search was employed to fine-tune the hyperparameter selection and more precisely identify the optimal solution. By reducing the range of values based on the results of the *Random Search*, we were able to more efficiently focus our computational resources on the most promising combinations of hyperparameters.

5. Performance Assessment

Following the completion of the model selection process, it was determined that *Adaptive Boosting and Bagging* with *Random Forest* as based estimator, *Stacking* and *Voting Classifier* with *Neural Networks, Random Forest* and *Support Vector Machines* as estimators and, lastly, *Random Forest* algorithm by itself demonstrated the most promising results (Fig. 17). *Decision Tree* on another hand decreased the performance with the parameters given by the *Random* and *Grid Search*, so we maintained the *Decision Tree* with the default hyperparameters.

To further optimize performance and reduce the discrepancy between the training and validation F1 scores, manual adjustments to the hyperparameters were implemented using values in close proximity to those selected by the *Grid Search*. Unfortunately, these efforts did not result in any noticeable improvement.

It is worth noting that we did not observe any signs of overfitting in the models. They were able to accurately predict all observations in the training dataset, achieving an F1 score of 1.0. When presented with the validation dataset, the models maintained a high level of performance, with an F1 score of 0.98. Given the small difference of 0.02 between the training and validation F1 scores, we are confident in the robustness of these models and consider them viable candidates for use in our final analysis.

In order to prepare the models to be exposed to the test data, we finally trained the models with all our dataset so it could have the best performance possible in the new data.

Using the models with the best performance as candidates, we submitted our results to Kaggle for evaluation. We selected Random Forest model as our final choice due to various reasons. First of all, it was able to have a final F1 score on Kaggle of 1.0. However, just relying on this proved to be unreliable, due to the fact that many models were able to reach the max score. The choice of the final model needed to be based on the model's consistency and our confidence in its reduced overfitting. Our choice with *Random Forest* was therefore due to its ability to stay consistent, highly precise and with high prediction capability through different choices of parameters and features. It also displayed consistent ability to improve the performance of other algorithms through its role as a base estimator. Summing up, even among models with a 1.0 F1 score, we have found this model to be the most effective through empirical evaluation.

6. Conclusion

As we progressed through the development of this project, we encountered a number of complex challenges that required us to utilize the Machine Learning knowledge and skills we had acquired through both self-study and classes.

We ended up developing our capabilities in data processing applying different techniques to deal with missing values and outliers, data analysis by creating plots that could help us to identify relationships between variables and see correlations and in application of models by understanding the principal concepts of each algorithm and the way they operate.

To predict which patients would be infected by the disease, we believe *Random Forest* algorithm would be the best approach since it gave an exceptional F1 score.

In conclusion, we learned that Machine Learning projects often involve an iterative process of trial and refinement. As a team, we developed a deeper appreciation for the complexities and challenges of this field and strengthened our passion for it.

7. References

[1]

"Random Forest | Methods | Differences | Real life applications | Data Science and Machine Learning," *Kaggle.com*, 2021. https://www.kaggle.com/discussions/general/208443

[2]

Gustavo Enrique Batista and M.-C. Monard, "A Study of K-Nearest Neighbour as an Imputation Method.," *ResearchGate*, 2002. https://www.researchgate.net/publication/220981745_A_Study_of_K-Nearest_Neighbour_as_an_Imputation_Method

[3]

P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," *arXiv.org*, 2019, doi: 10.48550/arXiv.1912.06059.

[4]

Zaher Al Aghbari, "Classification of Categorical and Numerical Data on Selected Subset of Features," *ResearchGate*, Aug. 18, 2010. https://www.researchgate.net/publication/221909332_Classification_of_Categorical_and_Numerical_Data_on_Selected_Subset_of_Features

[5]

Y. Meraihi, A. B. Gabis, S. Mirjalili, A. Ramdane-Cherif, and F. E. Alsaadi, "Machine Learning-Based Research for COVID-19 Detection, Diagnosis, and Prediction: A Survey," *SN Computer Science*, vol. 3, no. 4, May 2022, doi: 10.1007/s42979-022-01184-z.

[6]

"Passive Aggressive Classifiers - GeeksforGeeks," *GeeksforGeeks*, Jul. 08, 2020. https://www.geeksforgeeks.org/passive-aggressive-classifiers/

[7]

"Linear, Quadratic, and Regularized Discriminant Analysis," *Datascienceblog.net*, Nov. 30, 2018. https://www.datascienceblog.net/post/machine-learning/linear-discriminant-analysis/

[8]

V. Trevisan, "Target-encoding Categorical Variables - Towards Data Science," *Medium*, Mar. 17, 2022. https://towardsdatascience.com/dealing-with-categorical-variables-by-using-target-encoder-a0f1733a4c69

[9]

R. Wint, “On Maximal Information Coefficient: A Modern Approach for Finding Associations in Large Data sets,” *Medium*, Jan. 12, 2019. https://medium.com/@rhondenewint93/on-maximal-information-coefficient-a-modern-approach-for-finding-associations-in-large-data-sets-ba8c36ebb96b

Annex

1. Graphics and Tables

Variable Name	Type	Description
PatientID	Index	Unique patient identifier
Birth_Year	Numerical	Patient Birth-Year
Name	Categorical	Name of the Patient
Region	Categorical	Patient residence area
Education	Categorical	Category for the highest education degree
Disease	Target variable	Whether the patient is infected or not
Height	Numerical	Patient’s height
Weight	Numerical	Patient’s weight
Checkup	Categorical	How long has it been since visited a doctor for routine checkup
Diabetes	Categorical	Do the patient or their family have diabetes?
High_Cholesterol	Numerical	Patient’s cholesterol value
Blood_Pressure	Numerical	Patient’s average blood pressure
Mental_Health	Numerical	How many times, Patient’s poor physical or mental health has prevent them from doing usual activities in the past 30 days?
Physical_Health	Numerical	How many times, Patient’s poor physical health has prevent them from walking in the past 30 days?
Smoking_Habit	Categorical	Does the patient smoke more than 10 cigars daily?
Drinking_Habit	Categorical	What’s the patient behavior concerning alcohol consumption?
Exercise	Categorical	Does the patient exercise at least 3 times per week?
Fruit_Habit	Categorical	How many portions of fruits does the patient consume per day?
Water_Habit	Categorical	How much water does the patient drink per day?

Table 1 – Initial Variables

Variable Name	Type	Description
Male	Categorical (Dummy variable)	Patient's gender
		Patient's gender
age	Numerical	Patient's age
IMC	Numerical	Patient's Body Mass Index
is_fat	Categorical (Dummy variable)	Whether IMC > 25 or not
Diabetes_History	Categorical (Dummy variable)	Patient has/had pregnancy or borderline diabetes?
Diabetes_Family	Categorical (Dummy Variable)	Patient's family has diabetes?

Table 2 – Created Variables

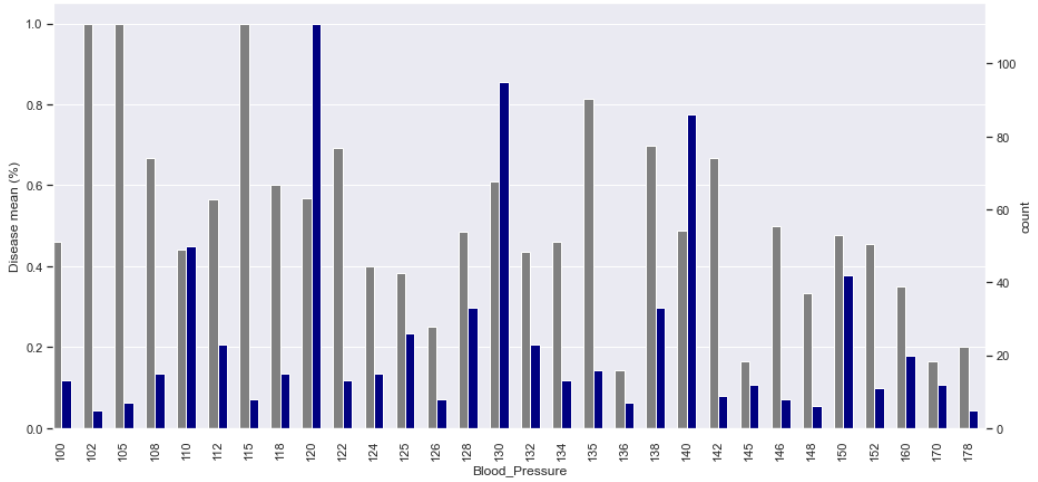

Figure 2 - Target means of top 30 values of 'Blood_Presssure' based on count

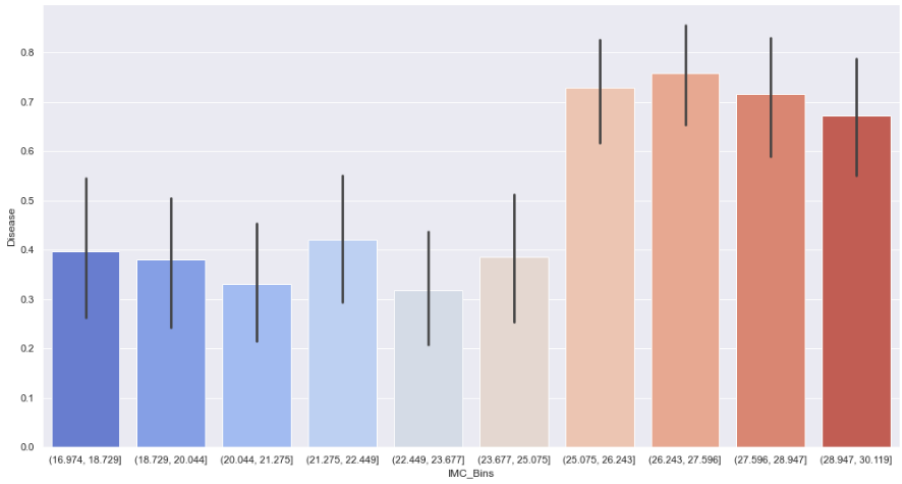

Figure 3 - target means of Binned 'IMC'

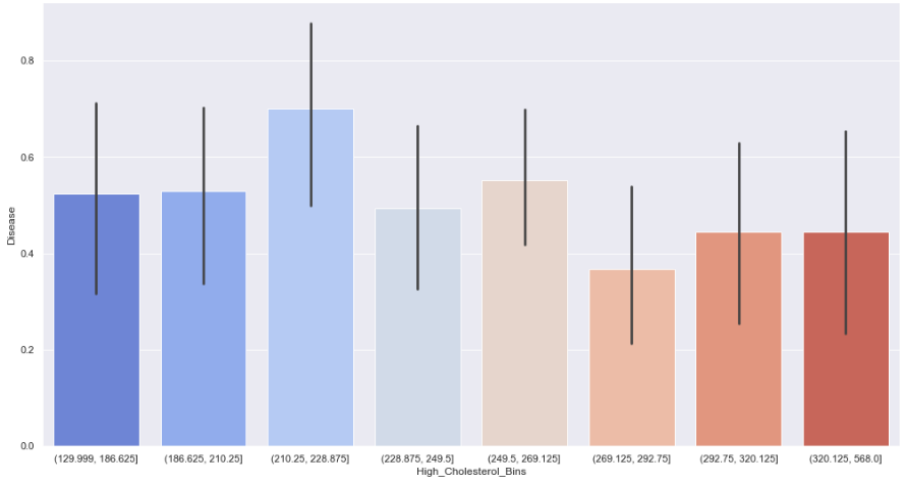

Figure 4 - Target means of binned 'High-Cholesterol'

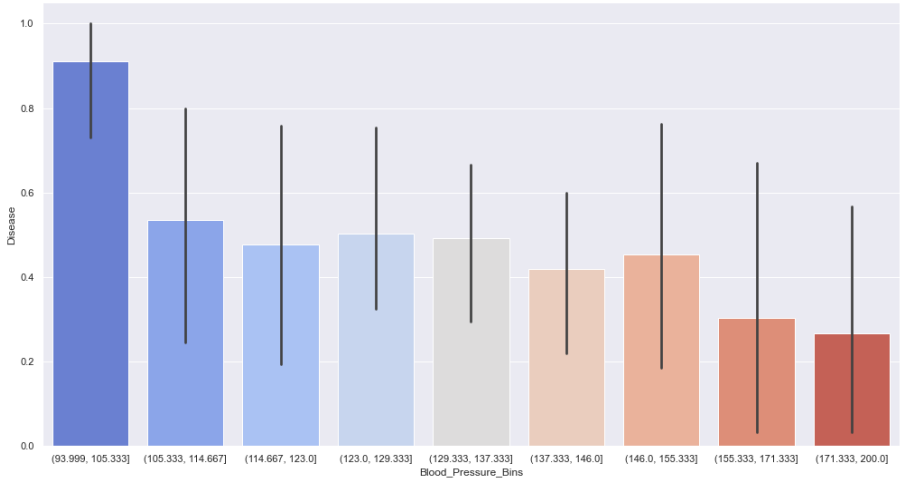

Figure 5 - Target means of binned 'Blood_Pressure'

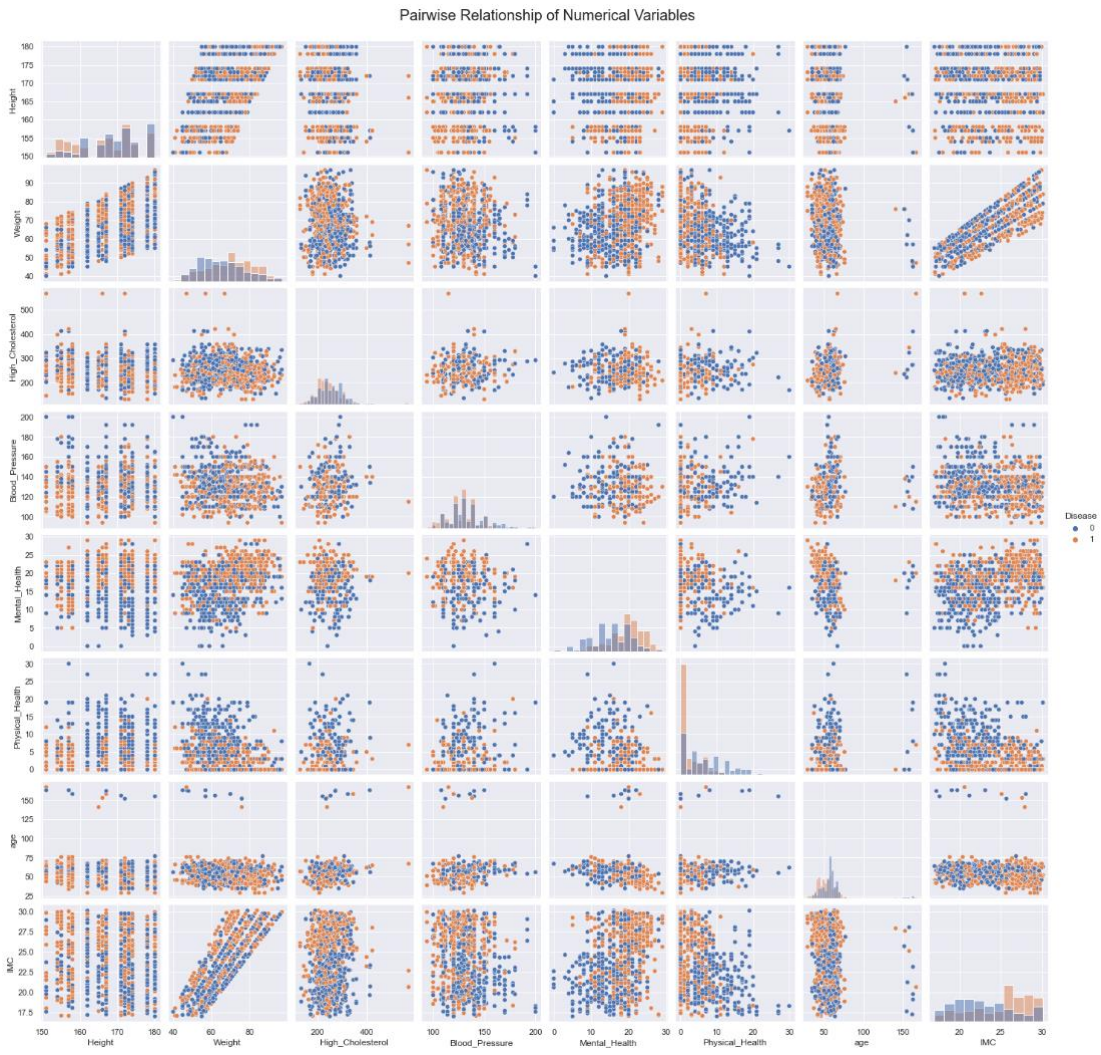

Figure 6 - Pairwise relationship of numerical variables with distinction based on Disease

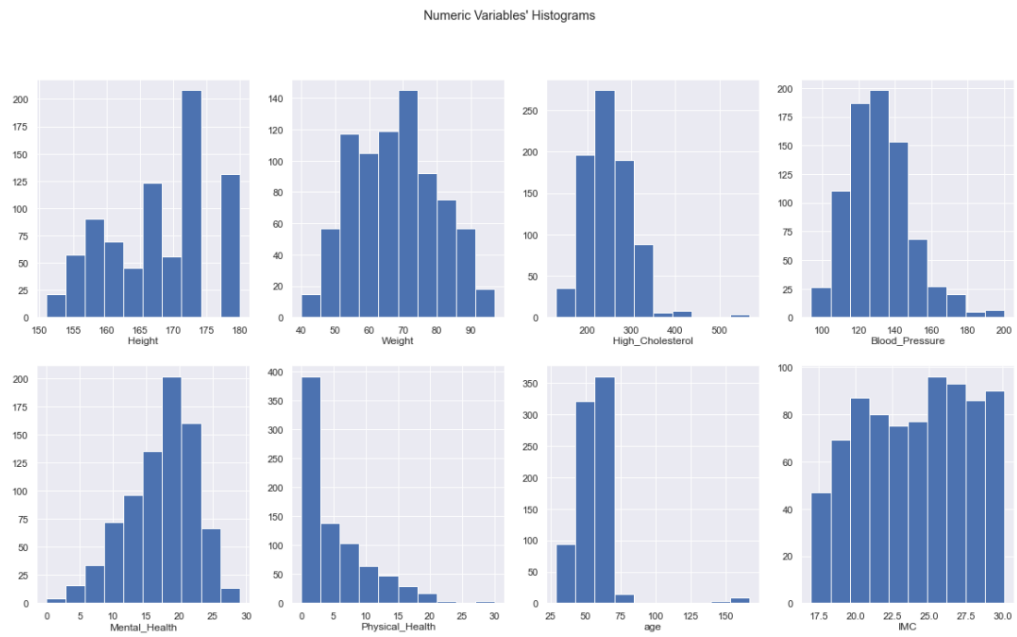

Figure 7 - Histograms pre-removal of Outliers

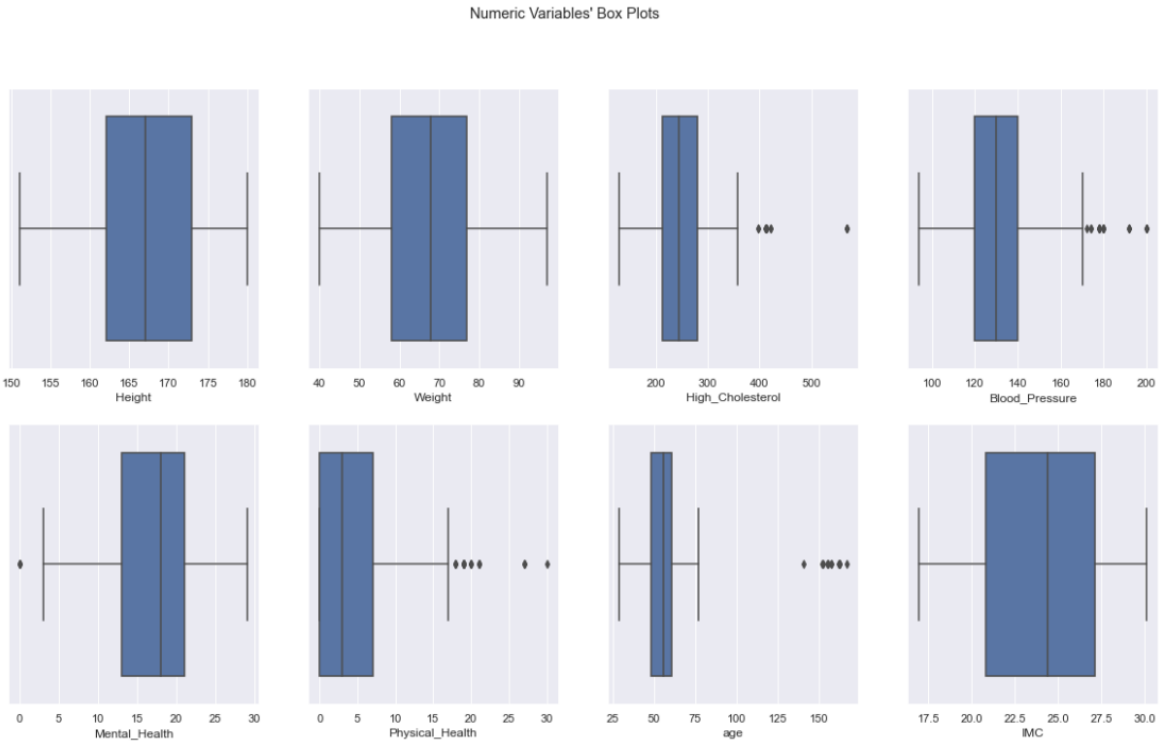

Figure 8 - Boxplots pre-removal of Outliers

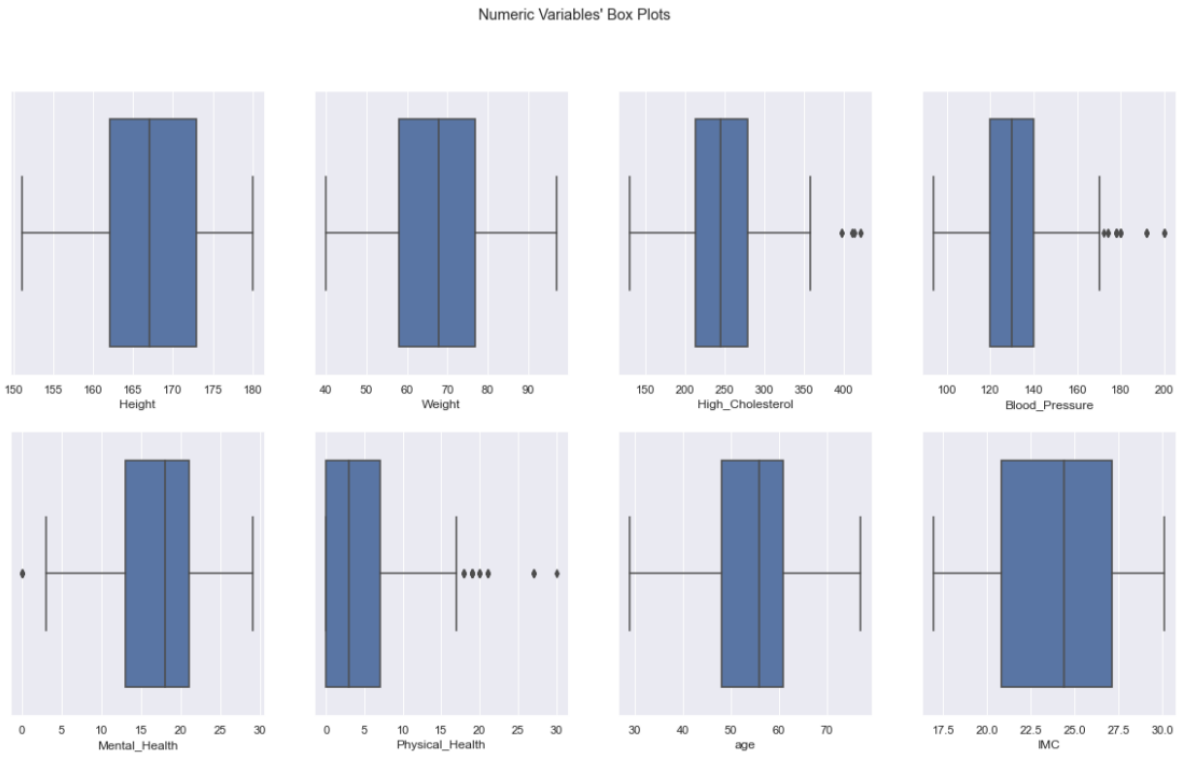

Figure 9 - Boxplots after-removal outliers

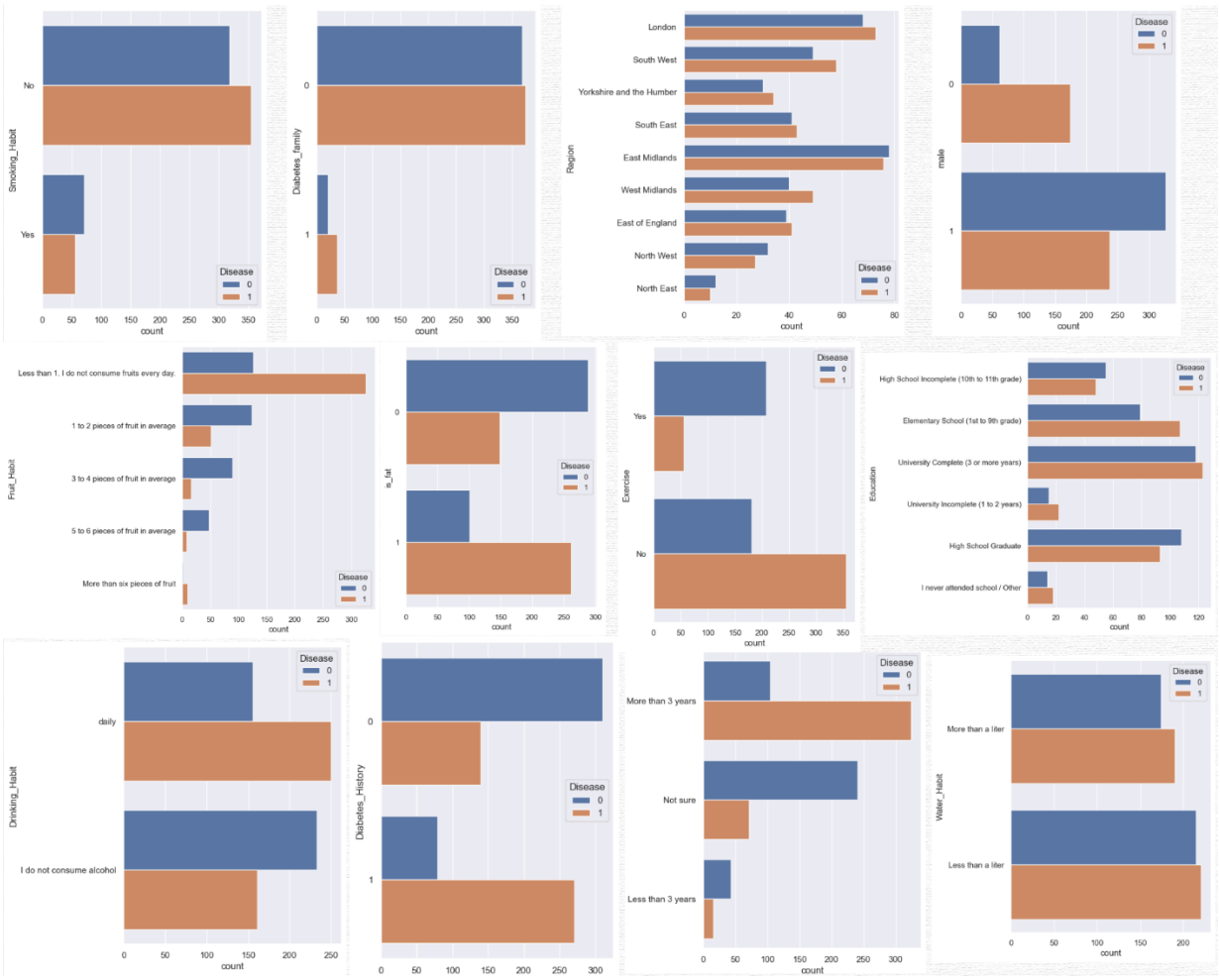

Figure 10 - Categorical variables with 'Disease' discrimination

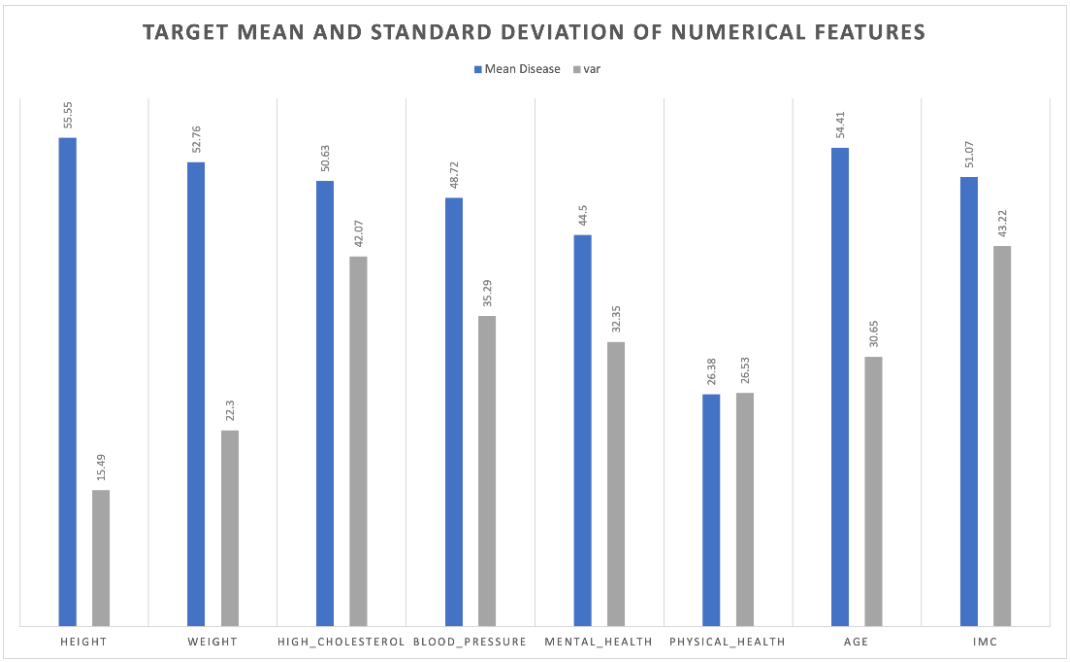

Figure 11 - Target mean and target standard deviation of target mean of numerical features

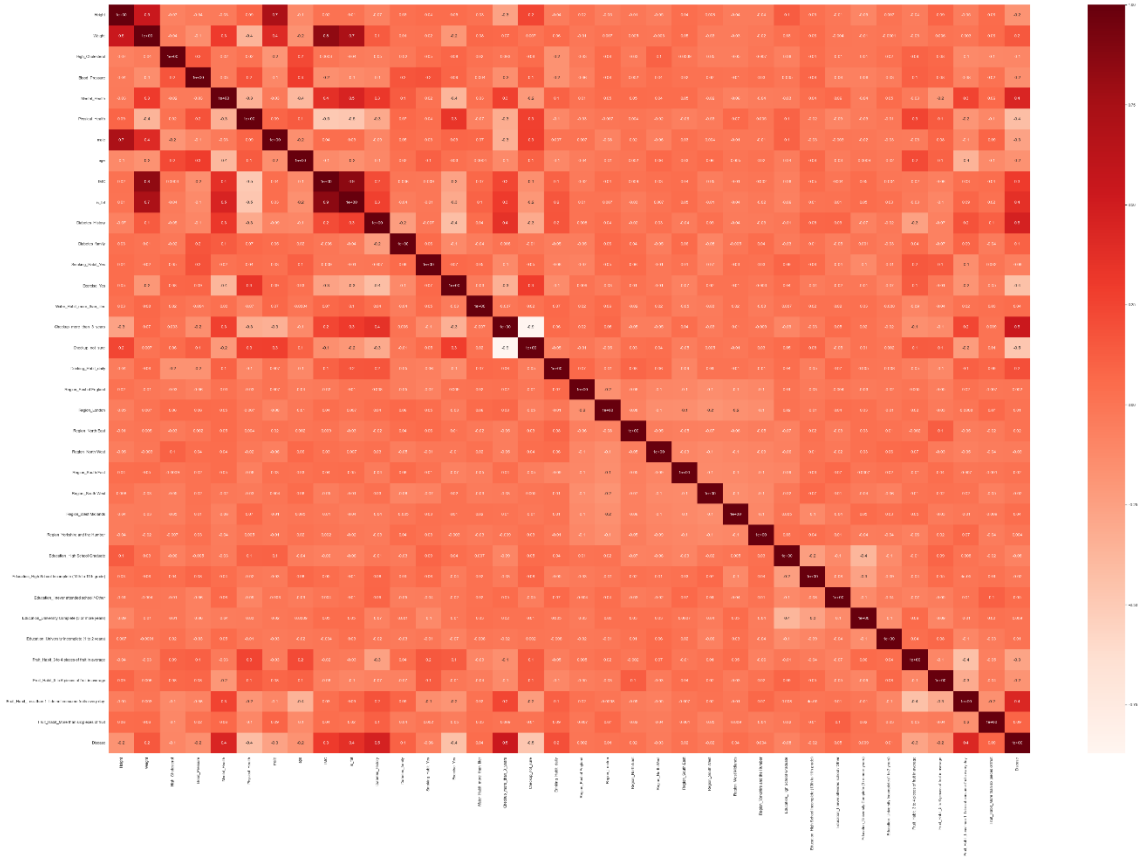

Figure 12 - Spearman Correlation Heatmap

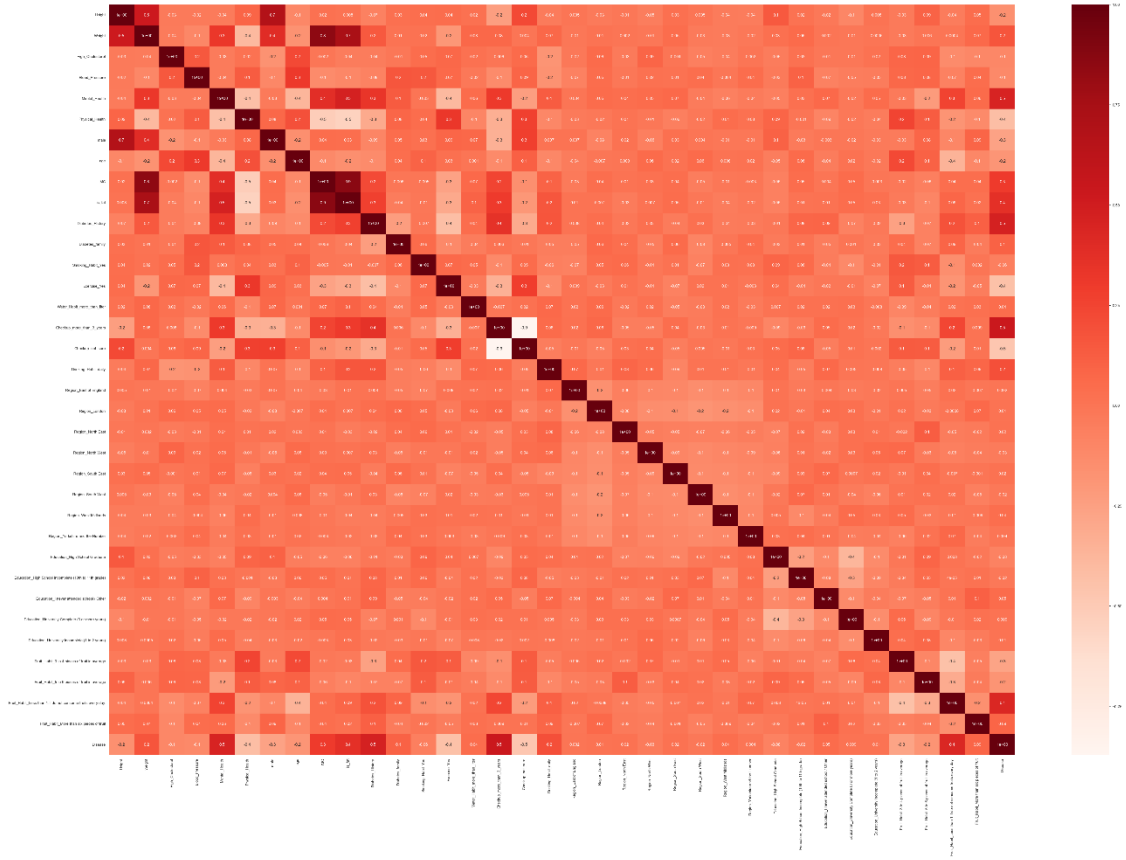

Figure 13 - Pearson Correlation Heatmap

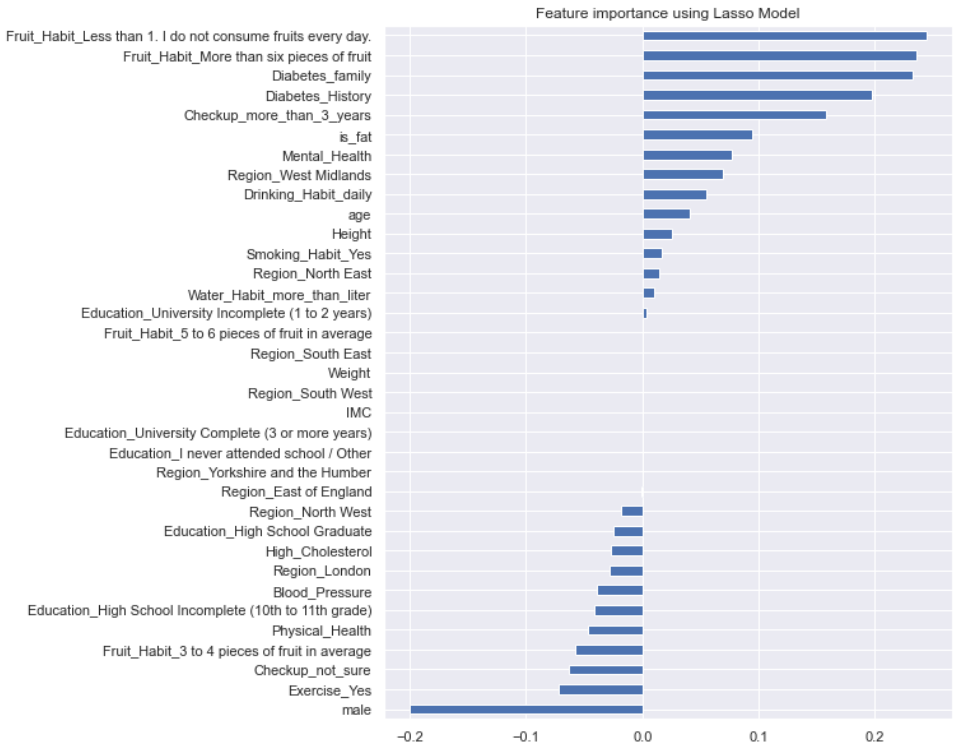

Figure 14 - Feature Importance - Lasso Regression

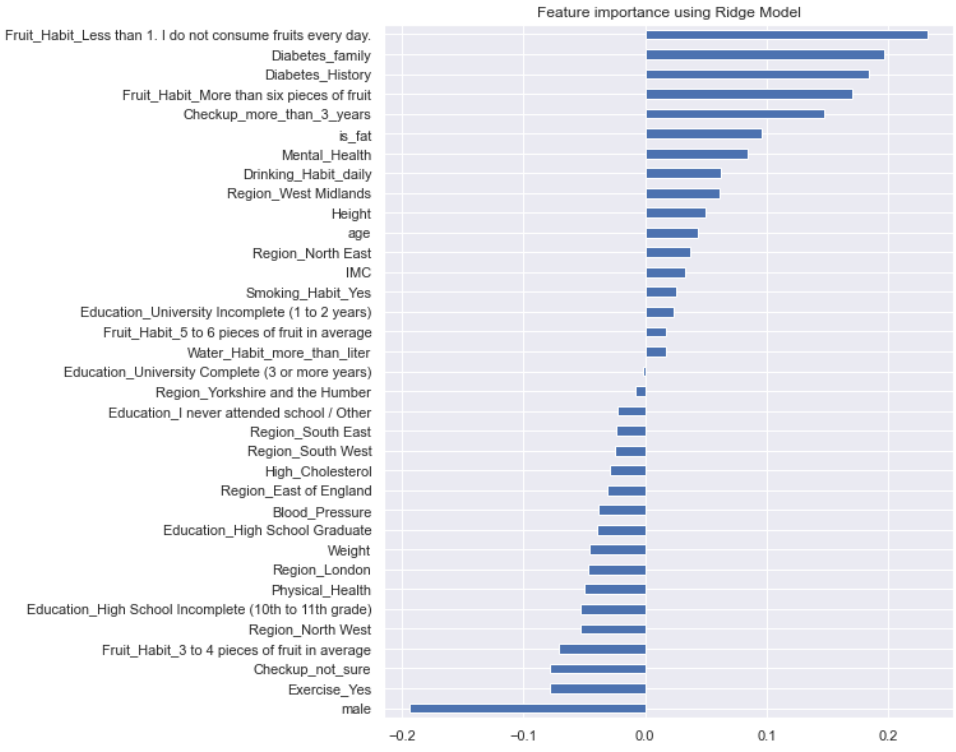

Figure 15 - Feature Importance - Ridge Regression

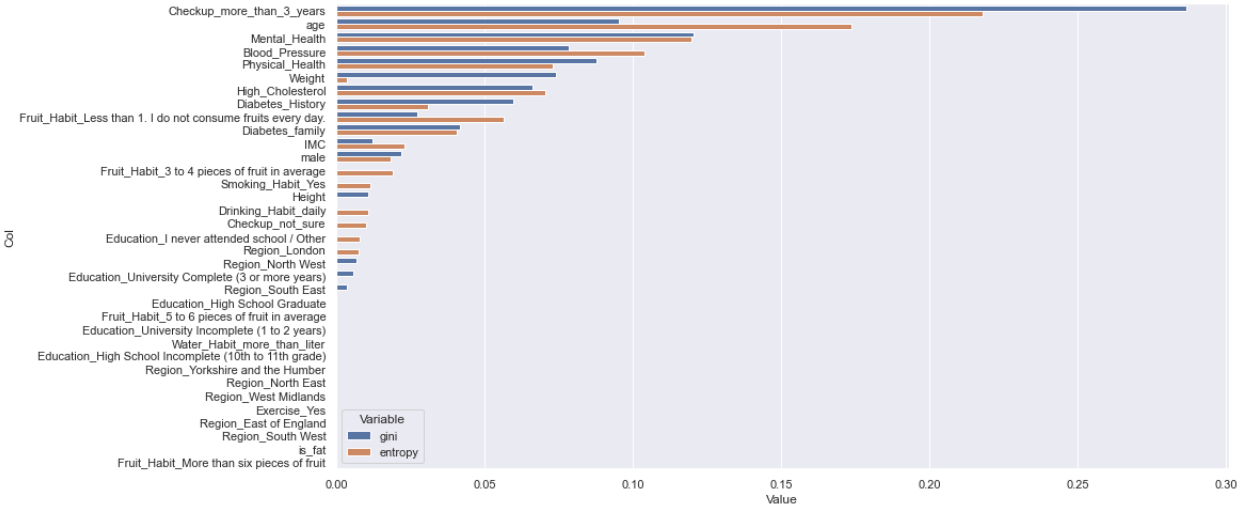

Figure 16 - Decision Tree for Feature Importance

	Filter Methods			Wrapper Methods				Embedded Methods			Final Decision
	Chi-Square	ANOVA	MIC	RFE	Forward Selection	Backward Selection	Stepwise Selection	Lasso Regression	Ridge Regression	Decision Tree	
Height											
Weight											
High_Cholesterol											
Blood_Pressure											
Mental_Health											
Physical_Health											
male											
age											
IMC											
is_fat											
Diabetes_History											
Diabetes_family											
Smoking_Habit_Yes											
Exercise_Yes											
Water_Habit_more_than_liter											
Checkup_more_than_3_years											
Checkup_not_sure											
Drinking_Habit_daily											
Region_East of England											
Region_London											
Region_North East											
Region_North West											
Region_South East											
Region_South West											
Region_West Midlands											
Region_Yorkshire and the Humber											
Education_High School Graduate											
Education_High School Incomplete (10th to 11th grade)											
Education_I never attended school / Other											
Education_University Complete (3 or more years)											
Education_University Incomplete (1 to 2 years)											
Fruit_Habit_3 to 4 pieces of fruit in average											
Fruit_Habit_5 to 6 pieces of fruit in average											
Fruit_Habit_Less than 1. I do not consume fruits every day.											
Fruit_Habit_More than six pieces of fruit											

Table 1 - Feature Selection

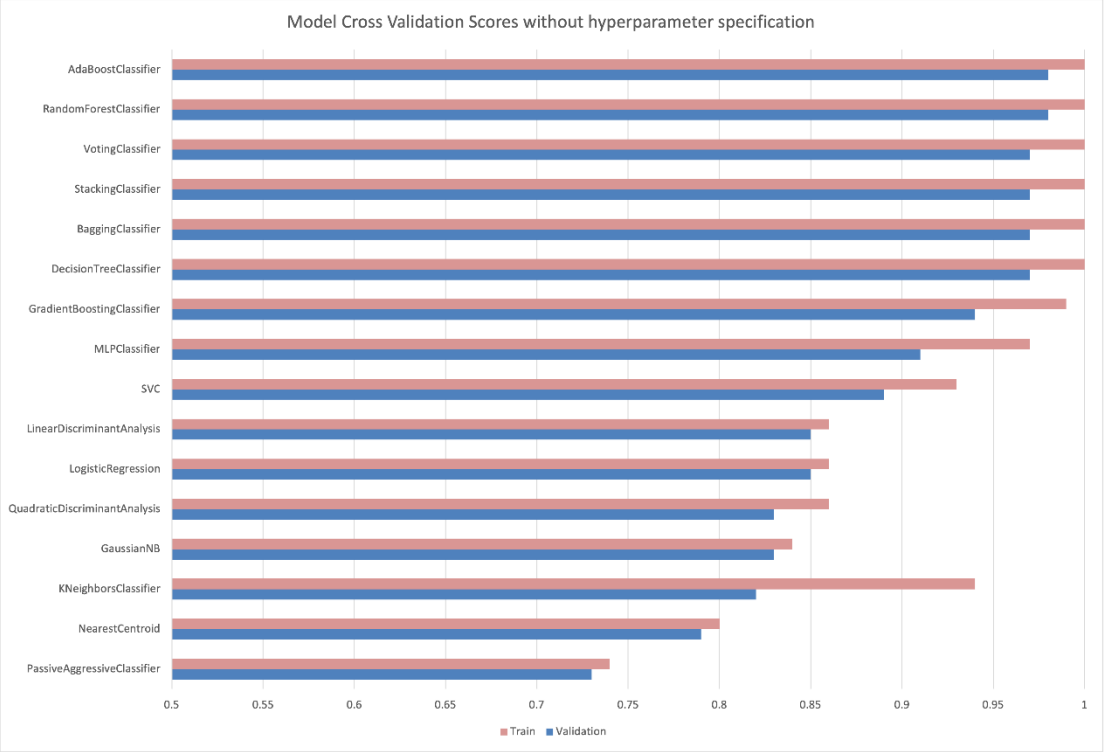

Figure 17 - Model scores with no parameters

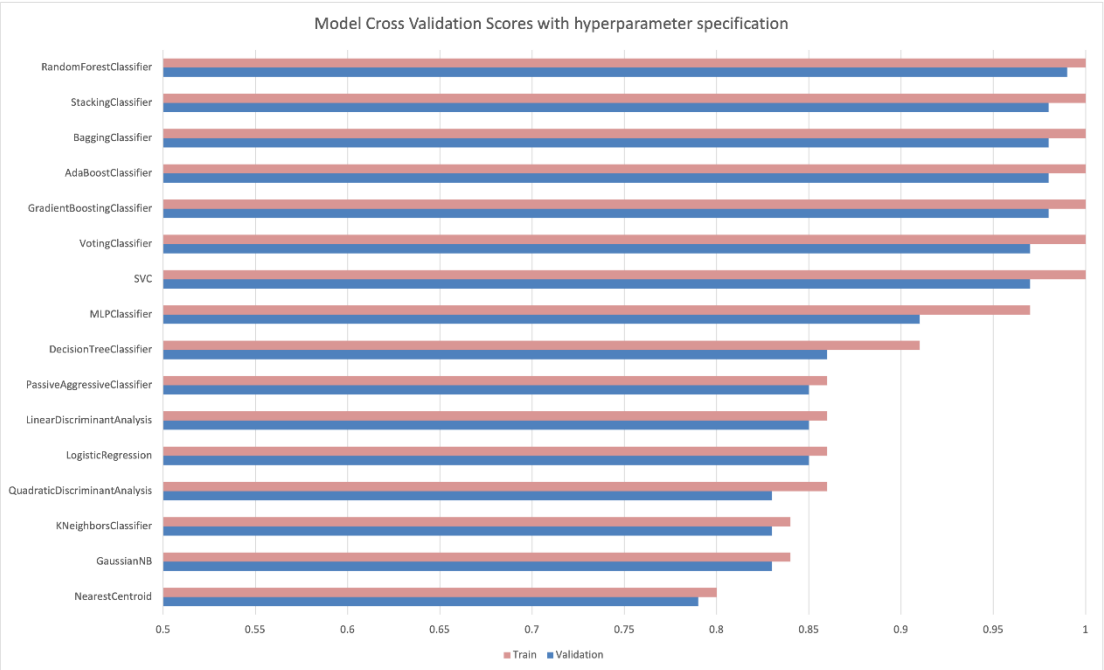

Figure 18 - Model scores with defined parameters

	Random Search Results - Best parameters	Grid Search results - Best parameters
DecisionTreeClassifier()	{'splitter': 'random', 'max_leaf_nodes': None, 'max_features': 6, 'max_depth': 6, 'criterion': 'entropy'}	{'splitter': 'random', 'max_leaf_nodes': None,'max_features': 8, 'max_depth': 6 ,'criterion': 'entropy'}
LogisticRegression()	{'solver': 'saga', 'penalty': 'l2', 'C': 100}	{'solver': 'saga', 'penalty': 'l1', 'C': 50}
MLPClassifier()	{'solver': 'adam', 'learning_rate': 'constant', 'hidden_layer_sizes': (10, 30, 10), 'alpha': 0.0001, 'activation': 'relu'}	{'solver': 'adam','learning_rate': 'constant', 'hidden_layer_sizes': (10, 30, 10), 'alpha': 0.0001, 'activation': 'relu' }
GaussianNB()	{'var_smoothing': 0.01}	{'var_smoothing': 0.01}
GradientBoostingClassifier()	{'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.1}	{ 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.1}
RandomForestClassifier()	{'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 110, 'bootstrap': False}	{'n_estimators': 250,'min_samples_split': 2,'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 110, 'bootstrap': False}
AdaBoostClassifier()	{'n_estimators': 100, 'learning_rate': 0.8}	{'n_estimators': 100, 'learning_rate': 0.8}
BaggingClasifier()	{'n_estimators': 100, 'max_samples': 0.8, 'max_features': 0.5, bootstrap=True}	{'n_estimators': 100, 'max_samples': 0.8, 'max_features': 0.5, bootstrap=True}
SVC()	{'C': 50, 'kernel':rbf }	{'C': 50, 'kernel':rbf }
KNeighborsClassifier()	{'n_neighbors':146 , 'p':1}	{'n_neighbors':146 , 'p':1
LinearDiscriminantAnalysis()	{'solver:lsqr}	{'solver:lsqr}
QuadraticDiscriminantAnalysis()	No hyperparameters defined	No hyperparameters defined
NearestCentroid()	{'metric':euclidean}	{'metric':euclidean}
VotingClassifier()	{'voting': hard}	{'voting': hard}
PassiveAggressiveClassifier()	{'C':0.001, 'fit_intercept': True, 'loss':squared_hinge, 'tol': 0.001}	{'C':0.001, 'fit_intercept': True, 'loss':squared_hinge, 'tol': 0.001}
StackingClassifier()	No hyperparameters defined	No hyperparameters defined

Table 2 – Best Parameters according to Random and Grid Search

2. Creativity and other self-study

K-Nearest Neighbor Imputer to treat outliers

In order to avoid the loss of information, we didn't want to simply drop the missing values and outliers in our data, so we decided to apply the KNN Imputer on these values. The KNN Imputer is a technique used to fill in missing data, taking into account the values of its closest neighbors. This method is beneficial, since it takes into consideration the values of other attributes, and not only the values of its column. To do so, we first had to turn all the outliers that we wanted to treat to NaN.

Since this method only works with numerical variables, and we also wanted to impute the missing values of the categorical variable 'Education', we first had to transform its categories into numerical ones, using the *LabelEncoder*, that attributes a number to each class. Then, we substituted the attributed value to the missing values by NaN, so that the *KNN Imputer* could compute these values, as well as the outliers previously replaced by NaN. Finally, after having the imputed values, we used the method inverse_transform to obtain the original categories of 'Education'.

Frequency Encoding

Frequency Encoding is a technique used in Machine Learning to convert categorical variables into numerical ones. This type of encoding involves replacing each category with the frequency of that category in the data. As an example, if Category A has 10 occurrences, then its frequency is 10 or 0.1, depending on the scale used in the encoding process. This type of encoding is very simple to implement and can be

effective in some cases, but it can also introduce bias into the model if the frequencies are not representative of the underlying data relationships.

Target Mean Encoding

Target Mean Encoding is also a method for encoding categorical variables in a dataset. It involves replacing the categories in a categorical variable with the mean value of the target variable for that category. For example, if you have three categorical values ‘product’ "x", "y", "z" and the target variable is ‘price’, to use *Target Mean Encoding* you calculate the mean price for each product and replace the respective products by the mean price of each one.

One advantage of *Target Mean Encoding* is that it can capture the relationship between a categorical variable and the target variable. It can also be useful when the categories in a categorical variable have a natural ordinal relationship, such as "low", "medium", and "high".

However, this method can also have some drawbacks. It can be sensitive to outliers, as a single outlier can significantly affect the mean of a category. It can also lead to overfitting if the number of observations for each category is small. In such cases, it may be better to use another encoding method, such as one-hot encoding or ordinal encoding.

Robust Scaling

Robust Scaling is a method for standardizing numerical data that is similar to the standard scaling, but more resistant to the influence of outliers. It scales the data by subtracting the median and dividing by the IQR, which has the effect of centering the data around the median and scaling it so that the IQR is equal to 1. This method is useful when the data contains outliers and it is desired to avoid distorting the data.

ANOVA

ANOVA, or *Analysis of Variance*, is a statistical test used to determine whether there are significant differences between the means of two or more groups. In Machine Learning, *ANOVA* can be used to identify which features have the greatest impact on the target variable. This technique of feature selection consists of the calculation of the F-statistic for each feature, comparing the variance between the target variable and the feature, to the variance within the feature. The interpretation of the output implies that the higher the F-statistic, the more impact the feature has on the target variable.

ANOVA can be used to improve the interpretability and accuracy of the model and it is also useful to reduce the number of features used in the model.

MIC

MIC, *Maximal Information Coefficient*, is a statistical measure used to identify the most informative variables in a dataset. In feature selection, *MIC* can be used to evaluate the relationship between a variable and the

target one by calculating the *MIC* value and selecting the features that have a stronger influence on the target variable, the ones with the higher *MIC* value.

MIC can be used to enhance the interpretability and accuracy of the model and additionally it is practical to lower the number of variables used in the model.

Forward, Backward and Stepwise Selection

Forward Selection starts with an empty set of features and adds one feature at a time, until the performance of the models stops improving or reaches a satisfactory level. On the contrary, *Backward Selection* starts with the entire set of features and removes one at a time, until the performance of the model stops improving or reaches a satisfactory level.

As a combination of both methods, Stepwise Selection starts with an empty set of features, and, as it is done in *Forward Selection*, adds features one by one, but it includes a step for removing features, like in *Backward Selection*. The process encompasses adding and removing features one by one until, again, the performance of the model stops improving or reaches a satisfactory level.

Ridge Regression

Ridge Regression is a linear regression technique that is used to address the problem of multicollinearity, which occurs when the features in the model are highly correlated with each other. This can lead to unstable and inconsistent coefficient estimates, which can make it difficult to interpret the results of the model. To address this issue, *Ridge Regression* introduces a regularization term to the objective function that is being optimized. This regularization term is the sum of the squares of the coefficients of the features in the model, and it is multiplied by a hyperparameter that controls the strength of the penalty. In *Ridge Regression* the magnitude of the coefficient reflects the strength of the relationship between the feature and the target variable. The features with a coefficient very close to 0 indicate that there is almost no correlation with the target variable, and thus it can be considered to drop them.

Decision Tree - Feature Importance

The *feature_importances_* method of a *Decision Tree Classifier* is a way to quantify how much each feature contributes to the decision-making process of the classifier. It returns an array of values that indicate the importance of each feature, with higher values indicating more importance.

For example, consider a *Decision Tree Classifier* that is trained to predict whether an individual has a certain disease based on their age, blood pressure, and cholesterol level. The *feature_importances_* method might return an array with values like [0.5, 0.3, 0.2], indicating that age is the most important feature, followed by blood pressure, and then cholesterol level.

Gini and *Entropy* are measures of impurity often used in *Decision Tree Algorithms*, that we considered when performing this technique. These are used to determine how pure a group of samples is, in other words, are used to determine which features to keep. The higher and similar *Gini* and *Entropy* values, the greater importance a feature has.

Gini is the probability of a randomly chosen sample from the group being misclassified if it is labeled according to the distribution of labels in the sample, whereas *Entropy* measures the amount of uncertainty in a group of samples.

Principal Component Analysis

Principal Component Analysis (PCA) reduces the dimensionality of a dataset, finding a new set of orthogonal uncorrelated variables, denoted by *Principal Components*, that capture the maximum amount of variance in the data.

After analyzing the eigenvalues and the cumulative variance, we concluded that our number of variables would come down to 10, which is where at least 80% of the variance is explained by the variables, and thus a small proportion of information is lost. Looking at the Scree Plot, we can see that the elbow of the graph is also around 10, which supports our decision of going with 10 PCs. Although in theory, this is a good method to reduce the number of features, the results when using Principal Components were always less accurate, so we decided not to use it.

	Eigenvalue	Difference	Proportion	Cumulative
1	2.907559	0.000000	0.244503	0.244503
2	1.543823	-1.363736	0.129824	0.374327
3	1.411309	-0.132514	0.11868	0.493007
4	0.957794	-0.453516	0.080543	0.57355
5	0.822746	-0.135047	0.069187	0.642737
6	0.587598	-0.235149	0.049412	0.692149
7	0.482238	-0.105359	0.040553	0.732702
8	0.382282	-0.099956	0.032147	0.764848
9	0.278141	-0.104141	0.023389	0.788238
10	0.264392	-0.013749	0.022233	0.810471

Table 5 - PCA

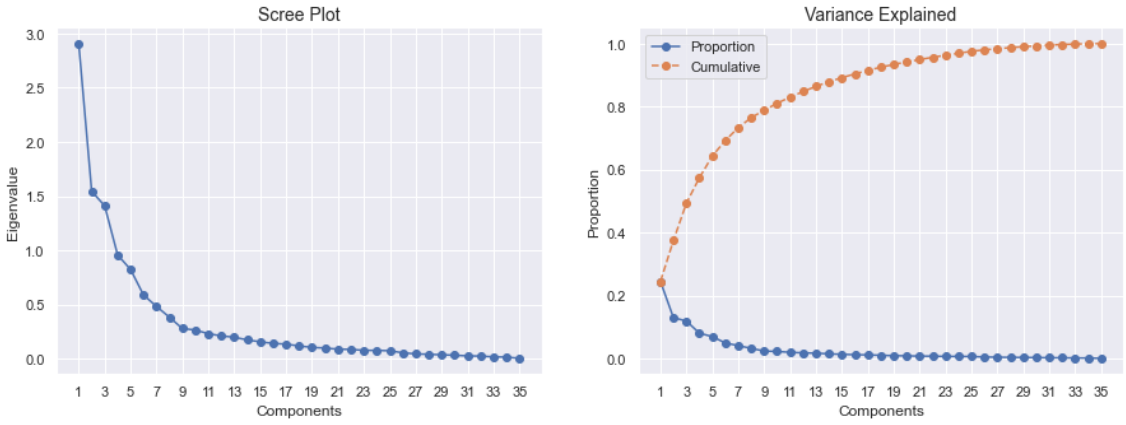

Figure 19 - PCA Graphs

Random Search

Random Search is a technique for optimizing the hyperparameters of a Machine Learning model. It works by sampling a set of hyperparameter values from a predetermined distribution, and then evaluating the model's performance using those values. This process is repeated a predetermined number of times, and the set of hyperparameters that yields the best performance is selected as the optimal configuration. This method allows us to quickly obtain a general understanding of the optimal hyperparameter values, enabling us to then fine-tune our models afterwards using more computationally intensive techniques

Combination of Random Search with Grid Search for tunning models hyperparameters

After a careful consideration, it was determined that the *Random Search* method would be the most suitable approach to use, first for optimizing the hyperparameters, due to its computational efficiency in contrast to the *Grid Search* method, which involves a thorough exploration of the entire hyperparameters space. *Random Search* was initially applied to a broad range of hyperparameters, the results of which were then used to generate a more precise and reduced set of hyperparameters to be evaluated using *Grid Search*. This hybrid approach allowed for a more targeted search for the optimal solution while minimizing computational demands.

Other Predictive Models

Passive-Aggressive Classifier

The *Passive-Aggressive* is a Machine Learning algorithm generally used for large-scale learning, that belongs to the online-learning algorithms. In this type of algorithm, the input data comes in sequential order and the model is updated at each step, contrary to what happens in batch learning, where the entire training dataset is used at once. It is especially helpful when there is a large amount of data and training the

complete dataset would be computationally impossible, due to the magnitude of the data. In simple terms, an algorithm for online learning will acquire a training example, update the classifier, and then discard the sample. The *Passive-Aggressive Classifier*, as the name implies, classifies instances as either “passive”, if the prediction is correct, or “aggressive”, if the prediction is incorrect, and thus the model needs changes.

K-Nearest Centroids

K-Nearest Centroids (*KNC*) is a classification algorithm that is used to divide a dataset into a specified number of clusters. It is a variant of the *K-Nearest Neighbors* (KNN) algorithm, with the main difference being that *KNC* uses centroids to represent each cluster, rather than individual data points.

One of the main advantages of *KNC* is that it is relatively simple to implement and understand, making it a good choice for beginners. However, it is not as accurate as some other classification algorithms, such as *Decision Trees* or *Support Vector Machines*, and may not perform well on datasets with complex patterns.

Overall, *KNC* can be a useful tool for clustering and classifying data, but it is important to consider the strengths and limitations of the algorithm when deciding whether it is the right choice for a particular task.

Voting Classifier

Voting Classifier is a type of ensemble classifier that combines the predictions of multiple base classifiers and uses a voting mechanism to make a final prediction. The goal of *Voting Classifier* is to improve the overall performance of the classifier by leveraging the strengths of different base classifiers.

The base classifiers used in *Voting Classifier* can be of different types, such as *Decision Trees*, *Support Vector Machines*, and *Neural Networks*. The final prediction is made by aggregating the predictions of the base classifiers using a majority vote or a weighted vote. In a majority vote, each classifier gets one vote and the class with the most votes are the one chosen as the final prediction. In a weighted vote, each classifier is assigned a weight based on its performance, and the class with the highest weighted vote is chosen as the final prediction.

Voting classifiers are commonly used in Machine Learning to improve the robustness and reliability of the classifier. They are particularly useful when the base classifiers have different strengths and weaknesses and can provide improved performance over a single classifier.

Linear Discriminant Analysis & Quadratic Discriminant Analysis

Linear Discriminant Analysis (*LDA*) and *Quadratic Discriminant Analysis* (*QDA*) are two types of linear classifiers that are commonly used in Machine Learning projects.

LDA is a supervised learning method that is used to classify data points into one of several different classes. It is based on the assumption that the data points within each class are normally distributed and that the

classes have the same covariance matrix. *LDA* works by finding a linear combination of the features of the data that best separates the different classes.

QDA is similar to *LDA*, but it relaxes the assumption that the classes have the same covariance matrix. This means that *QDA* can be more flexible than *LDA*, but it can also be more prone to overfitting if the data is not sufficiently large or if the number of classes is very large.

Overall, *LDA* and *QDA* can be useful tools for classification tasks in Machine Learning projects, especially when the classes are well separated, and the data is normally distributed. However, they may not be as effective when the classes are not well separated or when the data is not normally distributed. In these cases, it may be better to use a different type of classifier.