Machine Learning

Group Project 2022

"The Smith Parasite"

Professors:

Roberto Henriques

Carina Albuquerque

Ricardo Santos

Group Members:

Adriana Monteiro, 20220604

José Fernandes, 20220641

Pedro Ferreira, 20220589

Quintino Fernandes, 20220634

Index

1. Introduction

The Smith parasite, a disease named after its finder, Dr. Smith, has recently affected over 5000 people and has been the probable cause of aftereffects such as shortness of breath, loss of speech, confusion and chest pain.

The origin and method of transmission of the disease remain unclear, and many of the people infected present no symptoms. With that in mind, we have been tasked with finding underlying patterns that might indicate which groups are more prone to fall victim to this parasite through the creation of a predictive machine learning model.

Behavioural, health and sociodemographic information relating to a group of individuals, infected or not, has been shared with our task force and we have applied a multitude of data manipulation methods, as well as a variety of machine learning models, in order to gain the ability to accurately predict whether a given person is affected by the Smith parasite.

The training dataset is composed of 800 observations, with “Disease” being our target variable, which is binary. For any given row, a value of 1 for this variable means the observation corresponds to a person who got the disease, and a value of 0 means they didn’t. This dataset includes a variety of features, some of which we identified as categorical – “Name”, “Region”, “Education”, “Smoking_Habit”, “Drinking_Habit”, “Exercise”, “Fruit_Habit”, “Water_Habit”, “Checkup” and “Diabetes” – and others as numerical – “Birth_Year”, “Height”, “Weight”, “High_Cholesterol”, “Blood_Pressure”, “Mental_Health”, “Physical_Health”. Our goal is to optimize the performance of the best model we find based on validation and test f1-scores.

2. Exploration

Our analysis of the data started by joining the given data into to two distinct datasets, one for training and another one for testing. Setting the test dataset aside, we started by exploring the training dataset, checking the basic information of the observations. At first glance, the dataset seemed to have the correct datatypes for each feature and only the “Education” feature had 13 missing values.

By analysing the overall statistics of the training dataset, the histograms and the boxplots for each numerical feature (figures 1 and 2), we discovered that the oldest individual had a 138-year gap to the youngest. It happens that there was a group of outliers, in 12 specific observations of “Birth_Year” ranging from 1855 to 1881, while the rest of individuals were born only after 1945. Besides this feature, “Blood_Pressure”, “High_Cholesterol”, “Physical_Health” and “Mental_Health” also had outliers, all on the upper end of the data, with exception of “Mental_Health”, where 0 days corresponded to an outlier. We also did scatterplots (figure 3) comparing all metric features with one another and the only ones that seemed to have a slight correlation were “Weight” and “Height”.

In terms of categorical features, the biggest issue resided with “Checkup”, since almost 40% of the individuals were not sure when they had their last appointment. In other words, it’s as if these observations were missing values because they did not add any new information. In addition, while “London” and “LONDON” represented the same location, in “Region” they were considered two different categories, which we would correct later.

After this, we decided to plot each feature but comparing the individuals by the target label. The most apparent differences in the histograms (figure 4) were in “Weight”, “Mental_Health” and “Physical_Health” because individuals with the disease tended to have a higher average weight, a “worse” mental health and, contrary to what would probably be expected, reflected a better physical health (these last two in the previous 30 days). For categorical features (figure 5), the most evident distinctions were in “Drinking_Habit”, “Exercise” and “Fruit_Habit” since in average ill individuals seemed to consume more alcohol, exercise less and eat less fruit when compared to healthy ones. “Diabetes” and Checkup” also showed interesting results. The first because the majority of not infected individuals did not have diabetes opposed to individuals with the disease, who were more spread out across all diabetes categories. And the second because it indicated that a big portion of individuals with the disease were not sure when their last medical checkup had been, so this might be a valuable information for the future.

The last step of this part was performing a DBSCAN[1] analysis to discover multivariate outliers. Since our dataset has 7 metric features, we used “MinPts” = 14. Following the results from the plot on figure 6 we used 32 for epsilon. As a result, we found 33 outliers of this kind.

3. Pre-processing

3.1. Incoherencies

The first incoherence we decided to correct was the one on the “Region” feature we talked about in exploration. So, we corrected the fully uppercase one to be “London”, because it was the most common value, was written like the other regions and in the test set every London region was written that way as well.

The other inconsistency on our dataset was related to the outliers on the “Birth_Year” feature. So, as we knew the values couldn’t be right, we thought they were typos, because apart from those 12 observations, the birth years made sense. We thought that the feature itself was still trustable, so our decision was to change the number 8 on these birth years to a 9, as the values started in 1855 and ended in 1881, so it would be impossible that they were instead 1755 to 1781, not solving the inconsistency, and impossible to be 2055 to 2081, because these people would not have been born yet. Furthermore, we did this approach because if we assumed these values were just plain random values input in our dataset, we would not be able to trust the rest of the observations for this feature, which we thought we could.

3.2. Missing Values

As we saw on the exploration, this dataset only had 13 missing values (at least, as “Not a Number”), all on the “Education” feature. We decided we could not delete the rows where this happened, as the only missing values were on this feature so that would be a waste of information. Being few missing values, it would not make sense to delete the feature either. So, we decided it would be best to impute. As we thought a mode imputer would be too simplistic, we decided that we would later use KNNImputer, to get a more dynamic range of values.

The “Checkup” feature, on the other hand, had “disguised” missing values. When we saw that 312 patients responded with “Not sure”, we started to think if that would be useful for our disease detection. Beforehand, we thought that checkup was a feature which would help us by a matter of “if more time passed from the last checkup the patient has gone to, the more likely it could be to happen something related to the disease”, but with almost half of the values being “Not sure”, we started to think of it as a way to know if the patient was sure or not about when they went to the doctor for the last time, which could indirectly tell us a little more about the individual. Because of that, instead of removing the feature (as it had almost half of the values missing) or imputing the values (almost half of the values imputed could mislead our models), we decided that from this feature we would do a new binary one, in which what mattered would be if the individual was sure or not about the last time they went to a doctor.

3.3. Outliers

To the outliers we found on “High_Cholesterol” and “Blood_Pressure”, we decided not to remove them manually, because the other features on these observations seemed mostly fine, and because the values on these patients for these features could happen in the real world. A similar approach was taken to the outliers on “Physical_Health” and “Mental_Health”, but this time because by having some outliers changed or removed in these features, which have a small scale of 0 to 30 days, we would lose some representation to some values in that range.

3.4. Feature Engineering

Using the title before the names of the patients (“Mr.” or “Mrs.” – there aren’t any “Ms.”), we created a feature called “Gender”, because this is a category that was not included in the dataset and could be useful to identify patients. Afterwards, we dropped the “Name” feature, because it was useless to discriminate observations.

With the weight and height, we calculated the body mass index (BMI) and added that as another feature, using its formula.

When repeating the DBScan[1] after feature engineering, we found out that we had less outliers in the end (18). These weren't blindly removed; they were just a factor to keep an eye on when modelling.

3.5. Encoding

From here, the pre-processing was not directly done on the training set, because we committed early to cross-validation as the method of model assessment, which meant we would not split the data into training and validation, so we could not fit the encoder on the training set to then transform both the training and validation – and doing it only on the training set and then do cross-validation would mean there would be data leakage. So, the next two sections (this one included) will be just about the way we decided to encode and scale, because after these there will be a section on how we did it so we could later apply cross-validation.

So, for encoding, as we figured we had mostly features which, while not having a direct order relationship, had some kind of "more or less" relationship inside the categories (in the sense that a person being, for example, in university has "more" education than one who finished elementary school), so in every categorical feature but the region we decided we would use an ordinal encoder. We did the categories manually so the encoder didn't just randomly assign numbers, to maintain some kind of order. We also included the binary features (which were Boolean before or just "yes or no" answers to a question) on this encoding because the encoder will, of course, also put 0s and 1s to their categories, so while not being ordinal, they still can be encoded with ordinal encoder.

For the Region feature, however, we used the One Hot Encoder – which transforms the variable into dummies - encoding, because there really is not an order, not even implicit, on regions.

3.6. Scaling

For scaling, only for the numerical features, we decided we would start by using the default MinMaxScaler.

3.7. Joining the Encoding, Scaling and Imputing

Because of the cross-validation, we would have to use a Pipeline[2] to be able to prevent data leakage on our validation folds, even in RFE for feature selection, so we had to use, in the beginning of the pipelines, a way to encode the categorical features, scale the numerical and impute values on education. So, we used a Column Transformer[3], so we could separate the things we had to do to our categorical (ordinal and region) and numerical features, but all in the same step of a pipeline. Using that, we essentially built a pre-processor which would be able to encode, impute and scale, each to its respective type of data, inside a pipeline, avoiding pre-processing data leakage. This pre-processor creation was generalized in a function, so we could create one quickly for future different models which used different features, requiring a different pre-processor.

3.8. Feature Selection

For feature selection, we tried several different methods so we would see their outcomes and what variables seemed to be important on every approach, using them later as the base variables for our models and what variables seemed to be important only on some approaches, therefore becoming variables we tested on every specific model to see whether or not they could improve that model's performance.

We checked for the existence of univariate metric variables and concluded there were none. With correlation, the features to exclude were based only on the fact that weight seemed to be, logically, very correlated with BMI. Next up, we utilized a mutual information analysis to look at predictive relevancy of the features relative to our target. There, we learned that high cholesterol, fruit habit and diabetes are some of the features with the highest mutual information value. We also learned that BMI has a considerably higher mutual information value than weight, meaning it might be a more suitable feature for utilization.

A number of other feature selection methods were then applied, such as a Decision Tree, a Chi-Squared test for the categorical variables, a Lasso Regression and a Recursive Feature Elimination (RFE). The chi-squared was made for different significance levels so we could see if the features would change. Nonetheless, we only used the usual significance levels (0.01, 0.025, 0.05). The RFE method was applied to a variety of models to get more varied information, since we had not yet funnelled into one single type of model for our project. Within the same models, we also tested the RFE not only with the ideal number of features, but also with the number of features which obtained a similar score to the best one.

In the end, we aggregated the results of all the method applied in a single table, figure 11a and b, where we could compare side-by-side the relevancy of the different features across a variety of selection methods. From these aggregated results, we selected the features that would be picked by every single selection method and called them the “sure” features – for easier future reference, meaning we were confident they would be relevant for just about any model we experimented with. These features included the “High_Cholesterol”, “Physical_Health”, “Fruit_habit”, “Diabetes” and “Checkup_sure”. We then selected a subset of features that were not relevant for every single model but still showed significant relevance across the board and called them “unsure” features, meaning they might prove useful or not depending on the model we use and we need to test them on a case-by-case basis. These features include “Birth_Year”, “Blood_Pressure”, “Mental_Health”, “Weight”, “Gender”, “BMI”, “Exercise” and “Drinking_Habit”. We then proceeded to discard the remaining features, as they did not seem likely the prove relevant for model improvement regardless of the model we decided on. We notice as well, with the exception of “High_Cholesterol”, that all “sure features” were mentioned in the exploration as having major differences between the two targeted groups.

4. Modelling

Our modelling approach, such as the model assessment, was almost identical for every model. First, we used the base model, with the features in the feature selection which we thought had high discriminative power by themselves, the ones we called “sure” features. Then, we did a loop which would join the “sure” features with all combinations possible of the “unsure” features, defined on the section before this one. So, in the end of this loop, we got which features we should add to the model for better score. After this, we tuned the hyperparameters, first by doing an individual analysis of each parameter, and then by doing a grid search or a random search[4], or both, depending on the training time of each model (this is the part that changes according to the model, so we’ll explain our choices for each one). In the end, we got our optimized models with the best features for it and with the optimized parameters, with a validation score obtained with Repeated Stratified K-Fold, with 4 folds (so we would have 200 observations as validation) and 20 repetitions (to get a reliable score). All of this was made with cross-validation, always using a Pipeline[2] to join the pre-processing with the modelling and prevent data leakage, as it was said earlier. This was where our function to create a custom pre-processor for a set of features came in handy, because each model had its best features.

4.1. Random Forest Classifier

By running the loop of all the combinations of “unsure” features with the base ones, we found that this model should use, beyond the “sure” features, the “Birth_Year”, “Gender” and “Drinking_Habit” to improve its score. Then, we moved on to the hyperparameter tuning, accepting the features above as the best features for this specific model. First, we checked most of the parameters’ influence individually, finding out that the ones with more importance to the model optimization were: the number of estimators – number of trees in each decision tree the random forest is composed by – which seemed to stabilize for 100 or more, with small changes in the cross-validation score; bootstrapping seemed to improve the model when compared to having it as False; the criterion – a mathematical function that measures the quality of a split – seemed to change the score as much as bootstrapping (“gini” or “entropy”); class weight – which gives a weight to each target class, so is a more important parameter on more unbalanced datasets - also looked like it had a small impact on the score; finally, the maximum number of features when looking for the best split inside the trees. Other parameters tested, like maximum number of samples when bootstrapping or out of bag score seemed to either drastically worsen the score when comparing to the default, or not change anything, so were not further considered.

So, because a grid search is very computationally expensive when using a Random Forest (and most algorithms, to be fair), the approach for this model was to do five Random Searches – much more efficient than even one grid search -, which should output some different results each time, but by doing it a number of times increases the probability of getting the best parameter combination. Because of all that, after doing the random searches, we did one final Grid Search, with the parameters that changed from one random search to the others, so we could be more sure of the final parameters to choose. Because we did an extra filtering with the random searches, this was a way smaller grid search, which took much less time than one would with all the different parameters. In the end, we arrived to a final Random Forest Classifier, with a "balanced" class weight, 120 trees on the forest, an "entropy" criterion, bootstrap on and "sqrt" for the maximum number of features. This model got a cross-validation score of 97.77% in the end.

4.2. Extremely Randomized Trees Classifier[5]

Because Random Forest got a good results, we decided to try a similar model, called Extremely Randomized Trees Classifier[5], or Extra Trees for short. We first found that for this model the best "unsure" features to add were "Blood_Pressure", "Mental_Health", "BMI" and "Drinking_Habit". For hyperparameter tuning, we saw that the most important parameters to later put on the random search were the exact same as in the Random Forest, because of the similarity between these algorithms. So, we did the same approach, with five random searches and a final grid search with the parameters got in the random search. In the end, we got a final Extra Trees model with the "gini" criterion, 130 estimators, "sqrt" for maximum features, no bootstrapping and no class weights. This model achieved a cross-validation score of 98.05%.

4.3. Gradient Boosting Classifier

Gradient Boosting required a bit more of optimization to reach the levels of the other tree-based algorithms, because initially it had a way worse score than the others in the beginning (default and "sure" features). However, as this is an algorithm which is more robust against overfitting, it was worth it to try. Adding to the "sure" features, we found that the best combination of "unsure" ones was "Birth_Year", "Blood_Pressure", "Mental_Health", "Gender", "BMI", "Exercise", "Drinking_Habit". Starting hyperparameter tuning, we realized that the default parameters were holding this algorithm back. So, as we had been doing, we started by individually changing them to see the effect on the score (note: the parameters with the same name as the ones in the latter tree-based algorithms have the same meaning).We saw that the ones which were worth introducing into our random searches procedure were: the number of estimators, which seemed to benefit from being higher than 200; the learning rate - which weights the contribution of each tree – was better from 0.5 to 1.5; the loss function seemed to change the score when we chose "exponential" instead of "deviance", but for the worse; the maximum depth – which limits the number of nodes in each tree – seemed better for values around 4 or 5, instead of the default 3; the subsample – fraction of samples to fit the base learners on – changed for the better around 0.75; finally, using an initialization model gave the performance a huge boost, especially when using Extra Trees for it. The other set of test parameters didn't seem to change performance or just decreased it when comparing to the default values (maximum features, criterion, warm start). So, running the 5 random searches and a later grid search with their results, we achieved a final model with Extra Trees to do the initial predictions, a subsample of 0.75, 250 estimators in total, a max depth of the trees of 4, a learning rate of 0.5 and an exponential loss function. In the end, this model got a 97.58% cross-validation f1 score.

4.4. Support Vector Classifier (SVM)

Once again we tested different combinations of the "unsure" features with the "sure" features to figure out what set of variables would add more to our model. This time around, "Weight", "BMI", "Exercise" and "Drinking_Habit" were the features that seemed to improve the accuracy of our model the most.

We then moved on to parameter fine-tuning. Once more, we started off by analysing each parameter individually, and through that process we concluded that "C", "kernel", "gamma", "coef0" and "class_weight" were the parameters that more significantly impacted the model's performance. In the context of SVCs, the parameter "C" corresponds to a strictly positive value representing the strength of the regularization of the model, meaning a higher "C" value corresponds to a stricter model where the margin of the hyperplane is smaller and classification mistakes are more penalized, and vice-versa. In this particular instance, the value we arrived at was 1000, meaning a relatively small margin. The "kernel" parameter defines the type of kernel function which will be applied to the data which, in our case, turned out to be "rbf", which stands for Radial Basis Function. In terms of the "gamma" parameter, which manages the kernel coefficient, the ideal value turned out to be "scale", which is the default value. For the "coef0" parameter, which manipulates the independent term of the kernel function, the value that wielded the best results was 10. Finally, "class_weight", which was the least impactful of the relevant parameters, corresponds to the weights given to each one of the output classes. In our case, the ideal value for it turned out to be "None", meaning all classes are given weight 1, as opposed to "balanced", where the weights are set in a way that is inversely proportional to class frequency.

Similarly, to the case of the Random Forest, due to the high computational requirements of this type of model, we utilized 5 random searches to narrow our parameter options, since a grid search with all the initial options available could have been too heavy. During this random search, all 5 searches indicated the same ideal value for the "gamma" parameter and, therefore, we assumed that value (which was the default) as correct moving forward, and excluded it from future searches. We then performed a grid search on the reduced list of possible parameter values, which returned the above-described set of values: "C" = 1000, "kernel" = "rbf, "coef0" = 10, "class_weight" = "None". With all of this optimization performed, we arrived at a final model cross-validation score of 92.66%.

4.5. K-Neighbors Classifier

Without any parameter tuning or feature selection, this model had a poor performance compared to the rest of the models. By running the usual loop we found that adding the features "Birth_Year", "Gender" and "BMI" improved its score by about 5%. So, we started the hyperparameter tuning by checking the parameters individually: n_neighbors – the optimal number seemed to be 1 with the model dropping in performance drastically if the number increased; weights - "distance" seems to give better results meaning that the closer neighbors have a bigger influence than the ones who are further away; algorithm – we found the same exact result for the 3 different algorithms; metric - "manhattan" was the best metric to use for distance calculator in this model.

Since this is not a very computationally expensive model, we chose run the GridSearch with all the parameters we wanted to test, which gave us three combinations of best parameters (same score), we chose to use the combination equal to the one we got when we tested the parameters individually since it was one of the best three results given by the GridSearch. So, in the end, we ended up with a K-Neighbors Classifier with a number of neighbors of 1, a "manhattan" distance and a "uniform" class of weight. This model got a cross-validation score of 96.45%.

We decided to try to run the loop of all the combinations of "unsure" features with the base ones with the tuned model again, which gave us a better result by changing the unsure feature "BMI" for "Drinking_Habit". With this change the model got a final cross-validation score of 97.75%, bringing it closer to the other models used.

4.6. Multi Layered Perceptron Classifier

Then we choose to build a neural network with sklearn's MLPClassifier (MLP). Running the "unsure features"' loop improved the initial validation score in 10% and added as new features "Birth_Year", "Mental_Health", "Gender", "Exercise" and "Drinking_Habit". As in with other models, we tested the parameters individually, this time accounting for both the validation score and overfitting, having found that for a single layered MLP, the best number of neurons was 100. We also experimented with several MLPs with two layers and the best combination obtained was of 25 neurons on the first layer and 20 or 25 on the second. As for activation functions – what transforms our inputs into outputs and adds non linearity to the model – using a ReLU or tanh function proved have the highest performance. The optimal learning rates' initializations – which optimize the update weight rate during backpropagation – were 0.01 and 0.001 (only used when the MLP solver is adam or sgd). Regarding the solvers, while lbfgs and adam performed the best, we did not immediately exclude sgd since we suspected that the low score could be related with the lack of optimization of the parameter momentum, in other words, the solution obtained at first might be a local minimum and the model stopped before finding the global one. To solve this, when doing a random search, we added 'adaptive' as a learning rate and a few values of higher momentum than the default one.

Since the parameters differ depending on the solver, we tested a costume random search with the best parameters found for each one five times, and only for sgd was necessary an additional GridSearch to finetune the parameters. Then we compared the three final models and found that the best one had as parameters: two layers with size 25, a learning rate initialization of 0.01, adam as the solver and activation function relu. The final cross-validation score was of 97.66%.

4.7. Bagging Classifier

We started by testing a standard Bagging classifier with both a Decision Tree, and a KNN as base_estimators. The decision tree performed a lot better, as it would have been expected, even when we tried the KNN with the tuning we did previously.

Using once again the best combination of features loop, we found that by adding the features: "Birth_Year", "Mental_Health", "Gender", "Exercise" and "Drinking_Habit" we could improve the score by about 1%. Then, we started the hyperparameter tuning by checking the Bagging Classifier parameters individually, as usual: n_estimators – we found the best number of estimators was in the interval of [100,200]; max_samples – the best results were within the interval of [0.8,1.0]; max_features – the maximum number of features – had the performance of the model stable after the value of 0.4 until the max of 1.0; bootstrap – the scores were about the same for True and False, slightly better for False; bootstrap_features – we reached the same conclusion as the bootstrap parameter; warm_start – the score is the same for True or False; oob_score – this parameter also had no effect on the score.

As this model is computationally expensive, we used Random Search with the best parameters found in the previous step and after we used the Grid Search with the parameters found in the random searches. So, the best parameters for the final model were: n_estimators = 200; max_samples = 1.0; max_features = 0.4; bootstrap and bootstrap_features = False. This gave us a final cross-validation f1 score of 98.01%.

After this, we checked to see if changing any of the parameters of the base_estimator, in this case a Decision Tree, would have any positive effect on the model, but this search was unsuccessful, any change to the parameters decreased the score of the Bagging model, so the default Decision Tree model was maintained.

4.8. Voting Classifier[6]

In the end, we wanted a good way to join the different predictions of every good model we optimized. So, we joined our best models and used the cross_val_predict[7] function from sklearn to get the validation predictions from each model in a 4 fold stratified cross-validation. We tested two different approaches of voting classifiers – hard voting, and soft voting. For the first one, we chose to use the optimized KNN, MLPClassifier and Extra Trees, because we only wanted a single tree classifier and the method requires an odd number of models. The final score was 97.49%. For soft voting, we chose only the MLPClassifier and Extra Trees models because the probabilities of KNN skewed the results due to being always either 1 or 0. The final score for this model was 97.59%. We did not include the SVC model in any of the voting classifiers due to its score being low compared to the others.

5. Assessment

Using the scores from all the 4 folds (from all 20 repetitions) from the cross-validation performed on each model, we constructed a visualization (figure 12) that allows us to do a side-by-side comparison of all our finalized models.

The public Kaggle score achieved was 1.0 for every model we arrived at, with the exception of the SVC which fell short of that result. However, this public score is based only on about 40% of the test data and, therefore, these results will likely not remain the same when the full test set is used instead.

The SVC model had by far the worst score, as well as a large variance, so we discarded that one right away. All of the other models showed similar scores, having more of a disparity in their variances, but the Extremely Randomized Trees Classifier and the Bagging Classifier (also decision tree-based) showed a slightly higher score, and the Bagging Classifier also displayed one of the smallest variances across all models. However, the Bagging Classifier, like various of the other models, produced much lower minimum score across its different folds.

After finalizing all individual models, we attempted to use a Voting Classifier to join different models and achieve even better results but, as it turned out, the performance dropped relative to many of our individual models. This drop in performance shows that, although our top models make few mistakes, these mistakes tend to occur in the same observations for the majority of these models.

6. Conclusion

Hence, after evaluating all the data describing the patient's characteristics and selecting the ones that were likely to be the best predictors for finding whether or not a patient is affected by the disease, we concluded that the Extremely Randomized Trees Classifier was the better-fitting model for this task. This model exhibited our highest final average F1 score, while being one of the most consistent across all folds of our cross-validation (it presented no outliers on the score values). It presented a higher minimum score than its toughest competitor, the Bagging Classifier, regardless of its larger variance. It is also worth mentioning that, for this endeavour, tree-based method seemed to be very good performers.

All of our final models, aside from the SVC, displayed a similar level of performance, and most tended to make mistakes around the same observations, indicating that perhaps a set of asymptomatic individuals might be the cause of the remaining inaccuracy.

In the end, our final Extremely Randomized Trees Classifier model boasted a validation score of 98.05% (with the Kaggle public test score of 100%, likely to go down as the private score is released), and we are confident that the product of our work can become a valuable asset in the process of combating the Smith Parasite and mitigating its damages over society.

References

1. “Random Search for Hyper-Parameter Optimization”, 2012 James Bergstra and Yoshua Bengio: bergstra12a.dvi (jmlr.org)
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
7. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.BaggingClassifier
9. Sander, J., Ester, M., Kriegel, HP. et al. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. Data Mining and Knowledge Discovery 2, 169–194 (1998). https://doi.org/10.1023/A:1009745219419
10. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
11. https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
13. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html
14. https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html#sklearn-compose-columntransformer
15. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.mutual_info_classif
16. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
17. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier
18. https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

19. https://www.baeldung.com/cs/random-forest-vs-extremely-randomized-trees

20. https://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning/

Annexes

Figures

Figure 1.

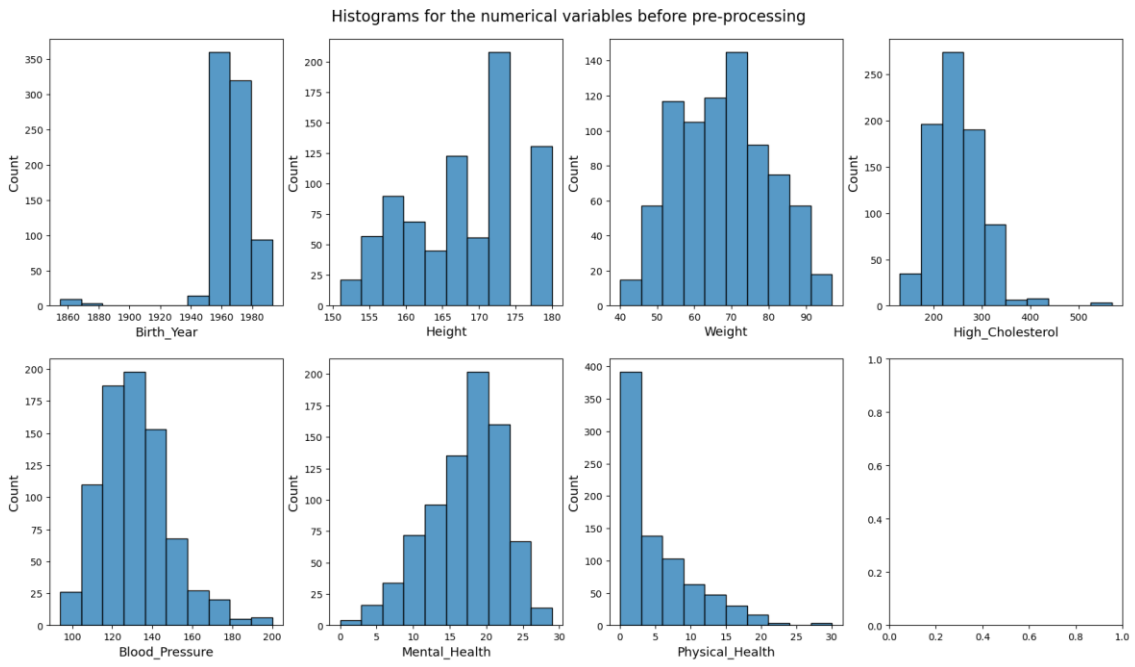


Figure 2.

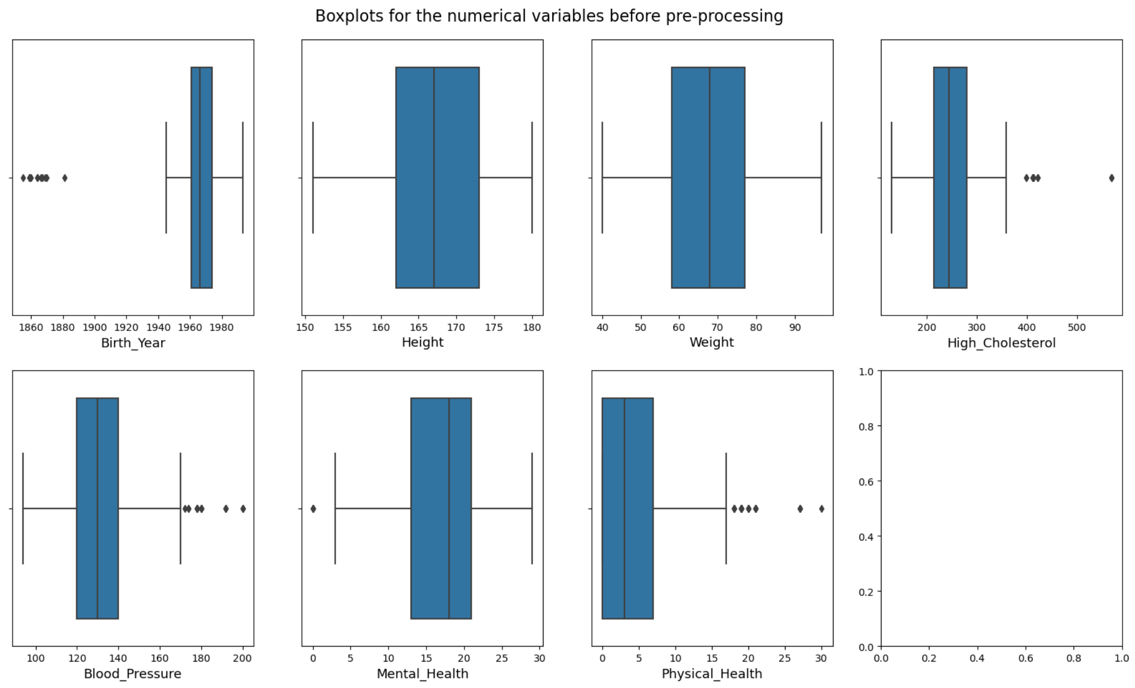


Figure 3. (Next page)

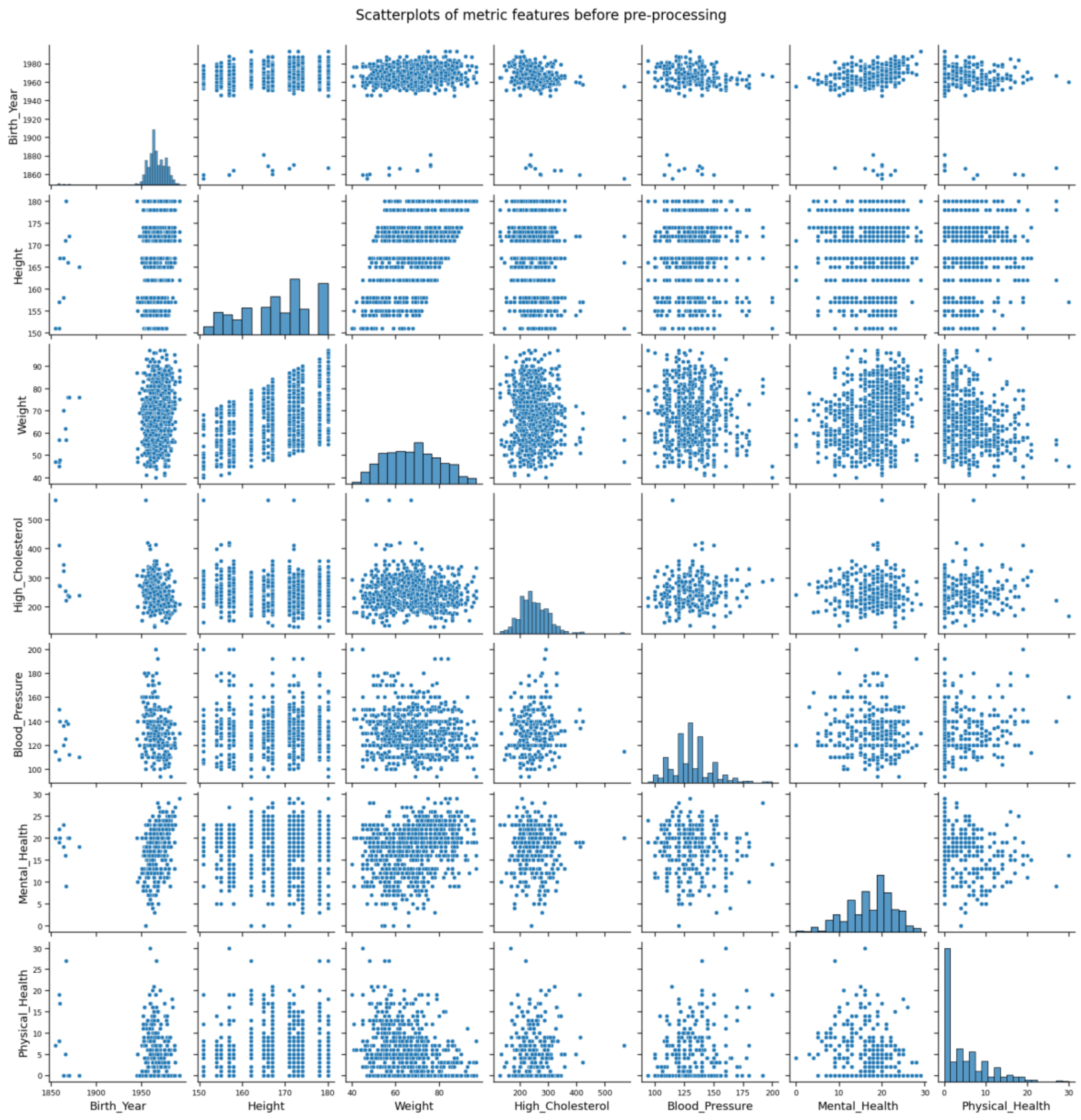

Scatterplots of metric features before pre-processing
Birth_Year
Height
Weight
High_Cholesterol
Blood_Pressure
Mental_Health
Physical_Health

Figure 4.

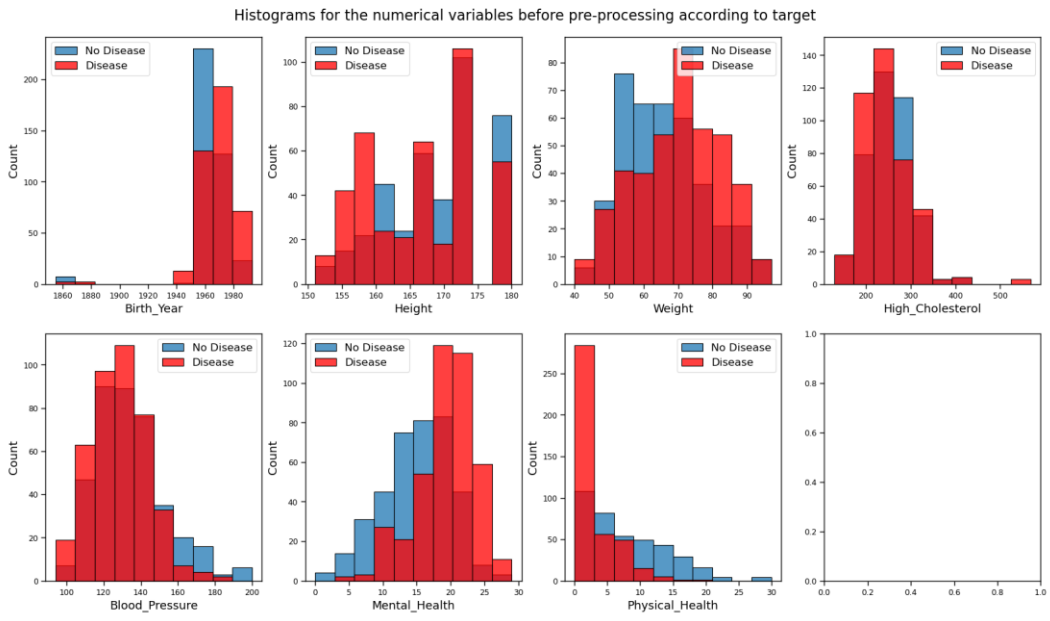

Histograms for the numerical variables before pre-processing according to target
No Disease
Disease
Count
Birth_Year
Height
Weight
High_Cholesterol
Blood_Pressure
Mental_Health
Physical_Health

Figure 5.: Some categorical features' barplots before pre-processing according to target:

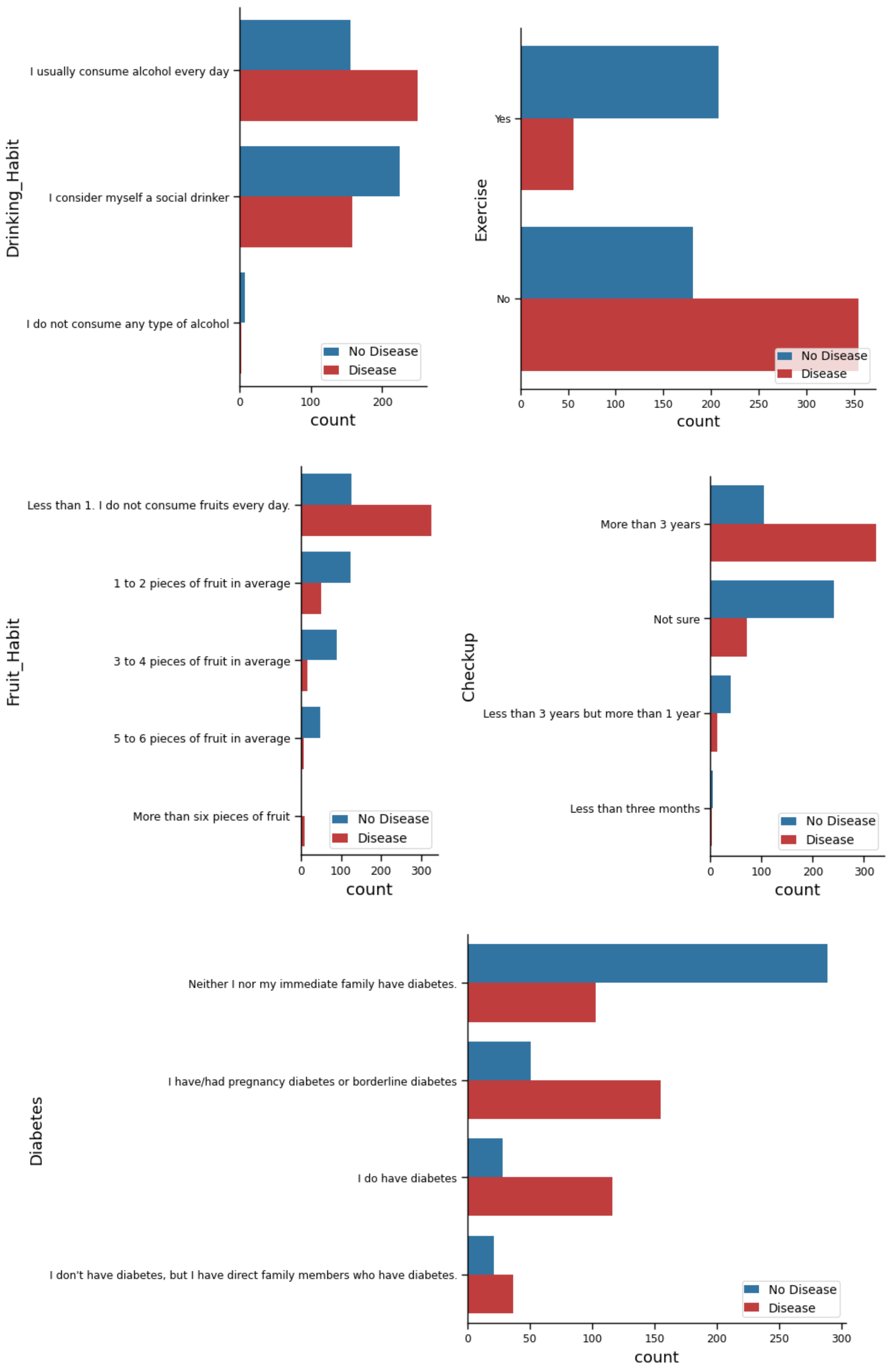


Figure 6.

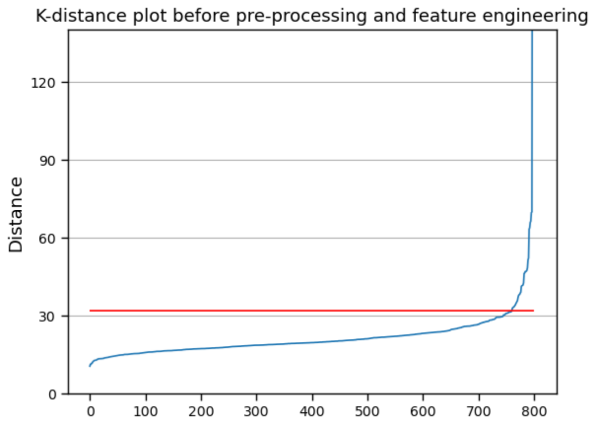


Figure 7.

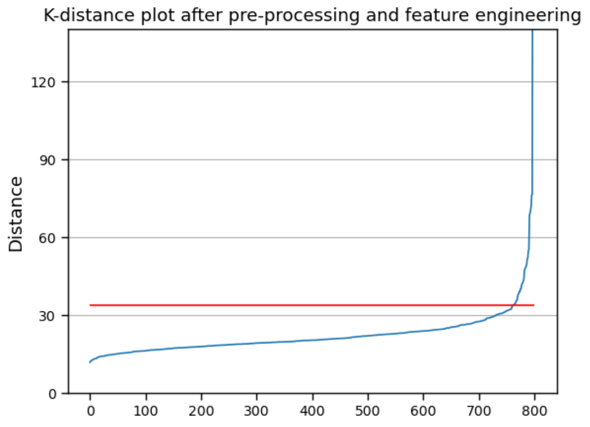


Figure 8.

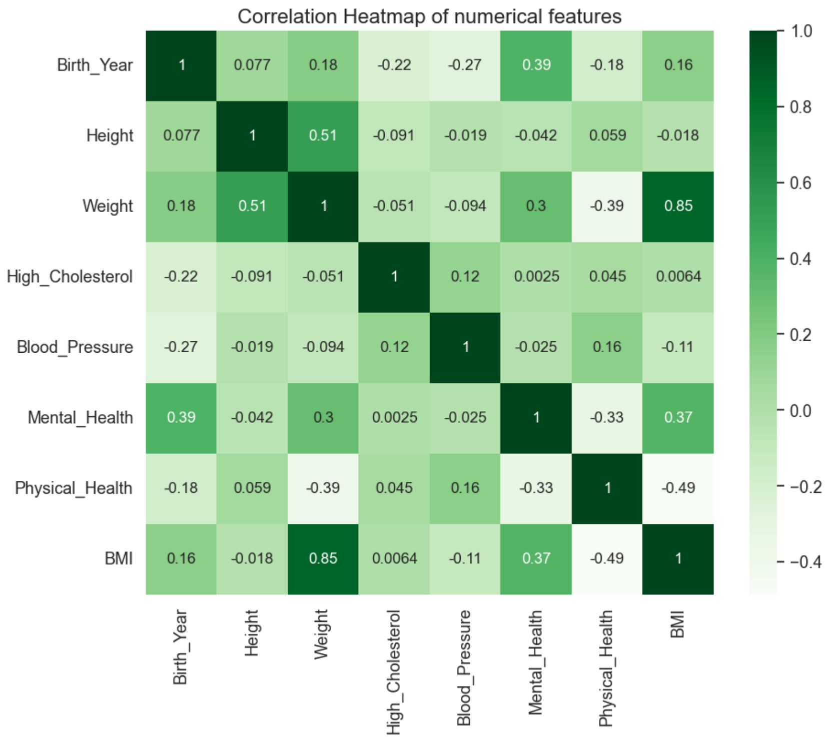


Figure 9.

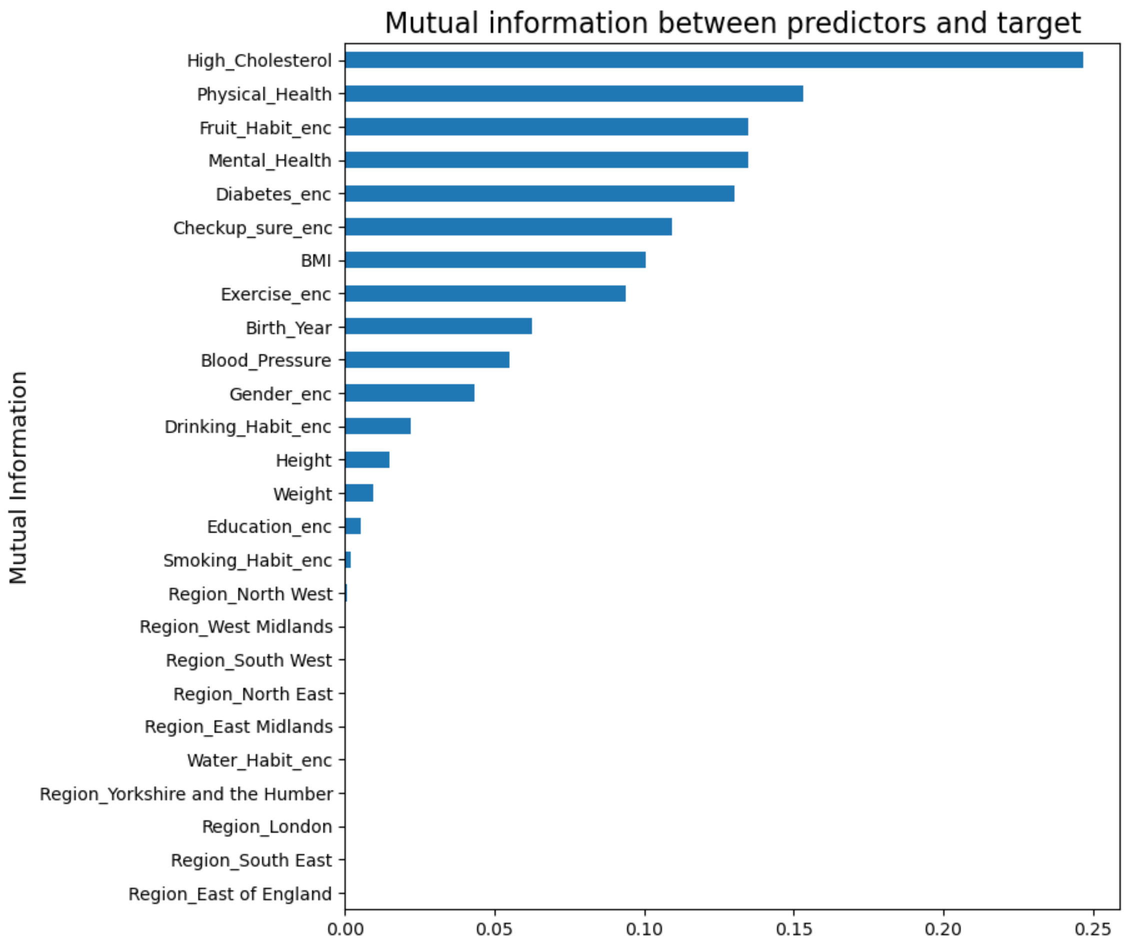
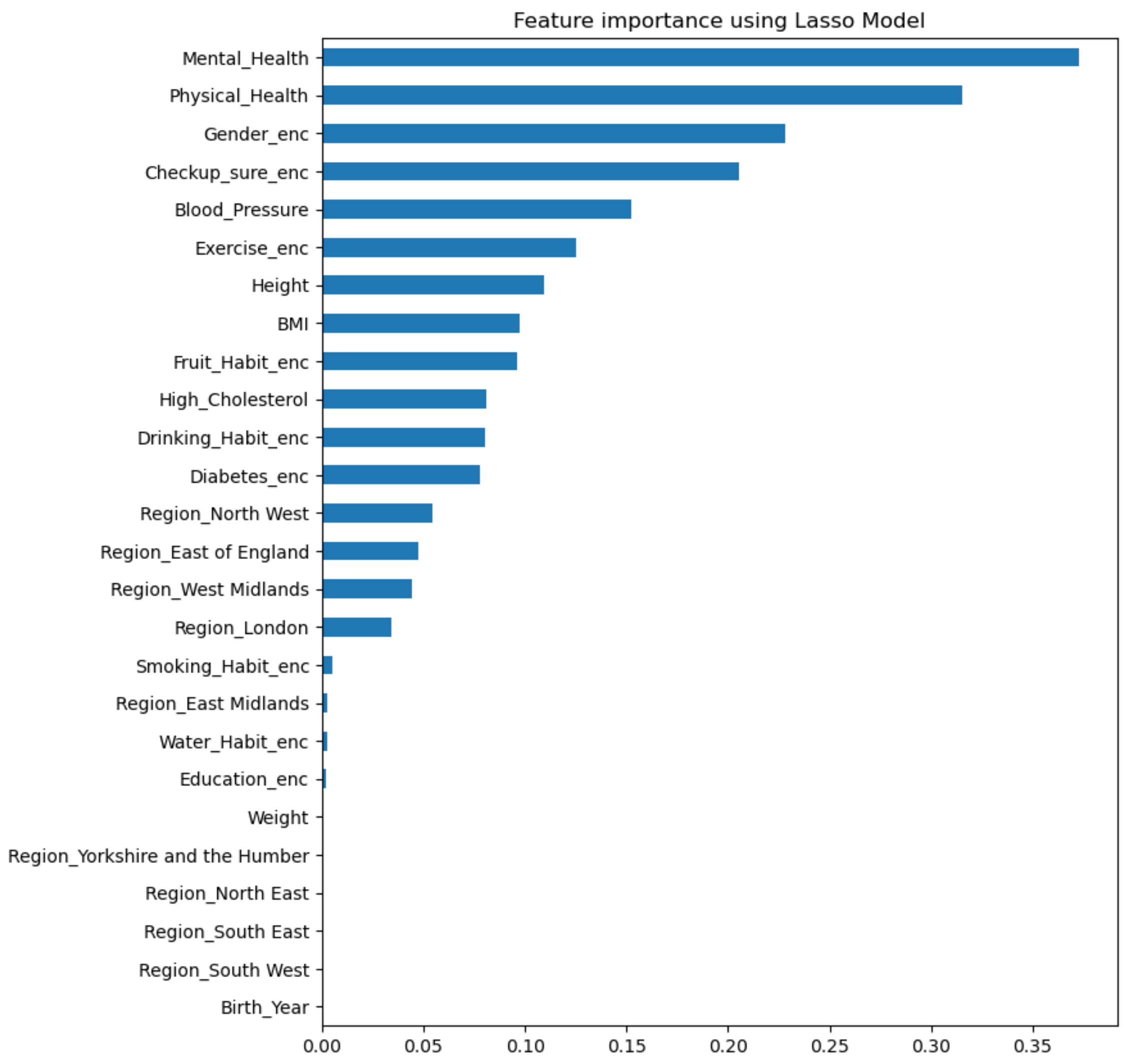

Mutual information between predictors and target
Mutual Information
High_Cholesterol
Physical_Health
Fruit_Habit_enc
Mental_Health
Diabetes_enc
Checkup_sure_enc
BMI
Exercise_enc
Birth_Year
Blood_Pressure
Gender_enc
Drinking_Habit_enc
Height
Weight
Education_enc
Smoking_Habit_enc
Region_North West
Region_West Midlands
Region_South West
Region_North East
Region_East Midlands
Water_Habit_enc
Region_Yorkshire and the Humber
Region_London
Region_South East
Region_East of England
0.00
0.05
0.10
0.15
0.20
0.25

Figure 10.

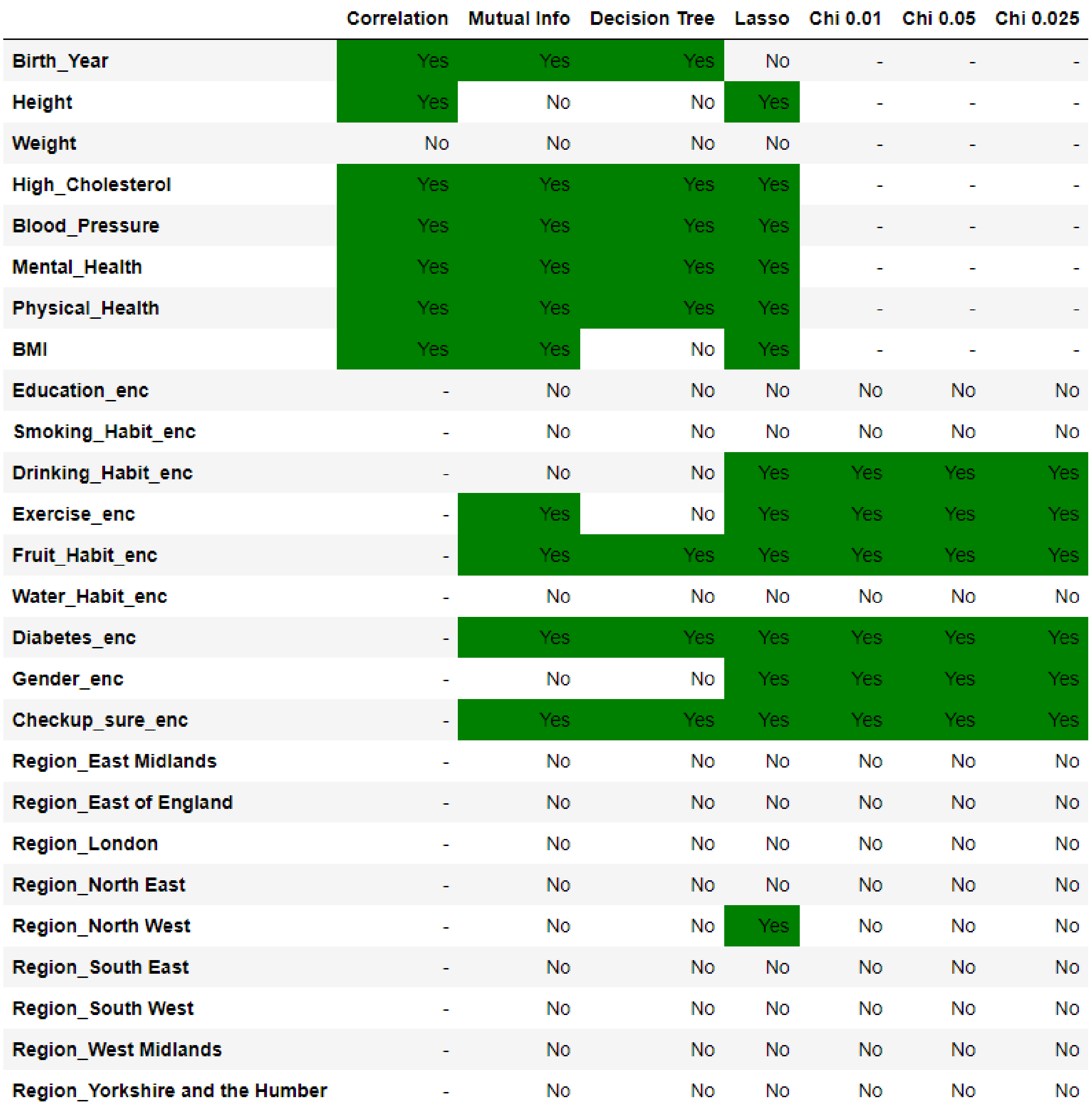


Feature importance using Lasso Model
Mental_Health
Physical_Health
Gender_enc
Checkup_sure_enc
Blood_Pressure
Exercise_enc
Height
BMI
Fruit_Habit_enc
High_Cholesterol
Drinking_Habit_enc
Diabetes_enc
Region_North West
Region_East of England
Region_West Midlands
Region_London
Smoking_Habit_enc
Region_East Midlands
Water_Habit_enc
Education_enc
Weight
Region_Yorkshire and the Humber
Region_North East
Region_South East
Region_South West
Birth_Year
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35

Figure 11.

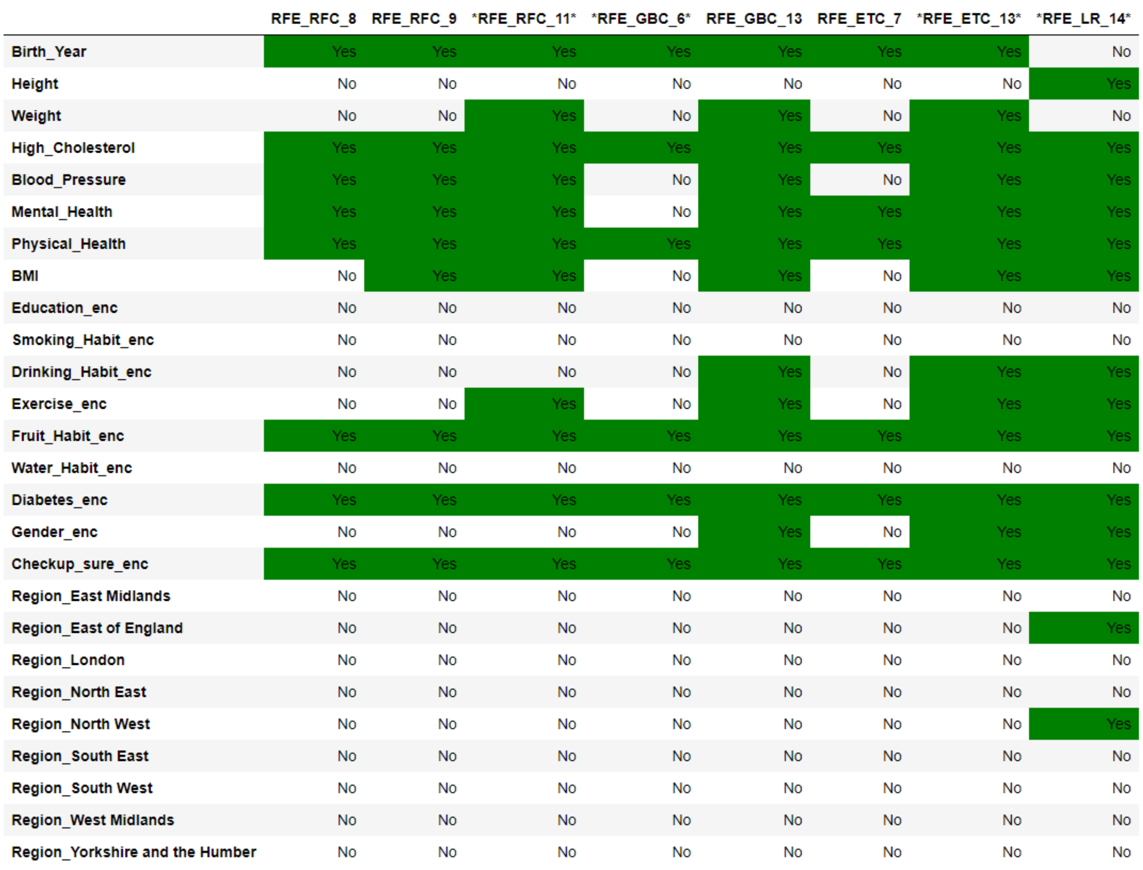
This figure shows two separate tables, where we aggregated every result from different feature selection algorithms. If “Yes” (and green), that model chose that feature, else, it did not.

a. (On the next page)
This figure shows the results for feature selection for, in the order of the columns, the correlation, mutual information, decision tree, lasso regression, chi-squared test for the usual significance levels (0.01, 0.025, 0.05).



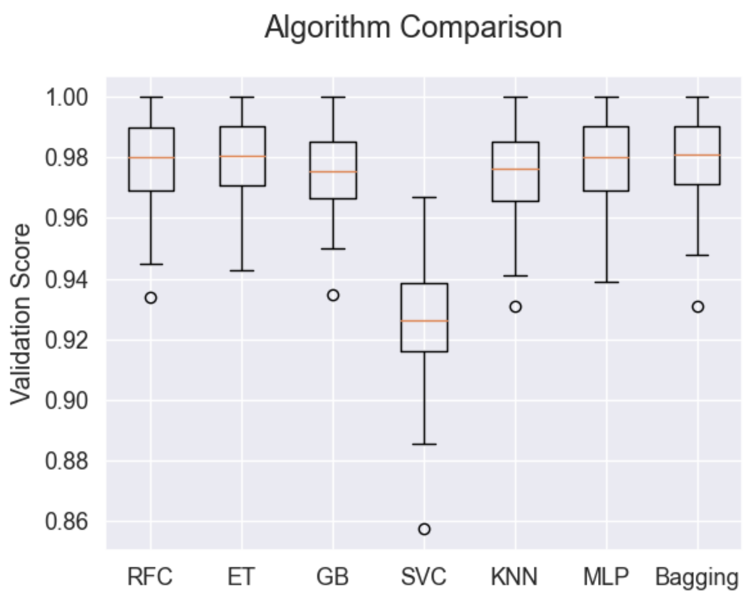
	Correlation	Mutual Info	Decision Tree	Lasso	Chi 0.01	Chi 0.05	Chi 0.025
Birth_Year	Yes	Yes	Yes	No	-	-	-
Height	Yes	No	No	Yes	-	-	-
Weight	No	No	No	No	-	-	-
High_Cholesterol	Yes	Yes	Yes	Yes	-	-	-
Blood_Pressure	Yes	Yes	Yes	Yes	-	-	-
Mental_Health	Yes	Yes	Yes	Yes	-	-	-
Physical_Health	Yes	Yes	Yes	Yes	-	-	-
BMI	Yes	Yes	No	Yes	-	-	-
Education_enc	-	No	No	No	No	No	No
Smoking_Habit_enc	-	No	No	No	No	No	No
Drinking_Habit_enc	-	No	No	Yes	Yes	Yes	Yes
Exercise_enc	-	Yes	No	Yes	Yes	Yes	Yes
Fruit_Habit_enc	-	Yes	Yes	Yes	Yes	Yes	Yes
Water_Habit_enc	-	No	No	No	No	No	No
Diabetes_enc	-	Yes	Yes	Yes	Yes	Yes	Yes
Gender_enc	-	No	No	Yes	Yes	Yes	Yes
Checkup_sure_enc	-	Yes	Yes	Yes	Yes	Yes	Yes
Region_East Midlands	-	No	No	No	No	No	No
Region_East of England	-	No	No	No	No	No	No
Region_London	-	No	No	No	No	No	No
Region_North East	-	No	No	No	No	No	No
Region_North West	-	No	No	Yes	No	No	No
Region_South East	-	No	No	No	No	No	No
Region_South West	-	No	No	No	No	No	No
Region_West Midlands	-	No	No	No	No	No	No
Region_Yorkshire and the Humber	-	No	No	No	No	No	No

b. (On the next page)
This figure shows the different results for the various recursive feature elimination for those different models. “RFE” stands for recursive feature elimination, “RFC” Random Forest Classifier, “GBC” is the Gradient Boosting Classifier, “ETC” Extremely Randomized Trees Classifier and “LR” the Logistic Regression. The numbers after the name of the model are the number of features for that RFE. If the column name is within asterisks (*name*), it’s the RFE for the best number of features we found for that model. We did with other numbers of features which had a really close score to the best one, hence the various numbers for one model.



	RFE_RFC_8	RFE_RFC_9	*RFE_RFC_11*	*RFE_GBC_6*	RFE_GBC_13	RFE_ETC_7	*RFE_ETC_13*	*RFE_LR_14*
Birth_Year	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Height	No	No	No	No	No	No	No	Yes
Weight	No	No	Yes	No	Yes	No	Yes	No
High_Cholesterol	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Blood_Pressure	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Mental_Health	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Physical_Health	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
BMI	No	Yes	Yes	No	Yes	No	Yes	Yes
Education_enc	No	No	No	No	No	No	No	No
Smoking_Habit_enc	No	No	No	No	No	No	No	No
Drinking_Habit_enc	No	No	No	No	Yes	No	Yes	Yes
Exercise_enc	No	No	Yes	No	Yes	No	Yes	Yes
Fruit_Habit_enc	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Water_Habit_enc	No	No	No	No	No	No	No	No
Diabetes_enc	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Gender_enc	No	No	No	No	Yes	No	Yes	Yes
Checkup_sure_enc	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Region_East Midlands	No	No	No	No	No	No	No	No
Region_East of England	No	No	No	No	No	No	No	Yes
Region_London	No	No	No	No	No	No	No	No
Region_North East	No	No	No	No	No	No	No	No
Region_North West	No	No	No	No	No	No	No	Yes
Region_South East	No	No	No	No	No	No	No	No
Region_South West	No	No	No	No	No	No	No	No
Region_West Midlands	No	No	No	No	No	No	No	No
Region_Yorkshire and the Humber	No	No	No	No	No	No	No	No

Fig. 12



Formulas

1. $BMI = \frac{weigth}{(heigth)^2}$

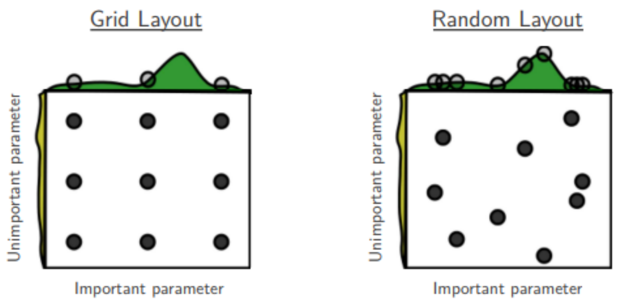
Concepts

1. DBSCAN is a clustering algorithm that identifies clusters as dense regions in the data space, in other words, groups together observations with many nearby neighbours, and marks the rest as noise points (outliers). It requires two parameters: $\mathcal{E}$ (*epsilon*): if any two points are at least $\mathcal{E}$ unities apart, they are considered neighbours and *MinPts*: the minimum of points in $\mathcal{E}$-radius. It is possible to finetune epsilon by plotting a k-distance graph, that is, the distance to the kth nearest neighbour (setting k as MinPts) and sorting these distances. This works because we expect that core points and border points' distance be much smaller than noise points (outliers). The outcome leads to a plot with an elbow that corresponds to the ideal epsilon value.

2. A pipeline is a function which sequentially applies whatever models/ transformers we put into it. It allows us to have a pre-processor inside it and then a model, for example, in a way which allows the cross-validation we use it on to not suffer from data leakage. So, when we put a pipeline inside a cross-validation, it will fit everything on the training folds, and only transform on the validation folds. Even if we just want to use a pipeline in a normal train-validation split fashion, it simplifies everything, as the pre-processing can be done in the same line of code as the modelling, granted we define what pre-processing we want to do in it.

3. A Column Transformer is a function which allows us to separate transformers for their corresponding sets of features in our mixed dataset. For example, if we have a set with numerical and categorical data and we want to have only one pre-processor for the whole dataset (again, very helpful for cross-validation with pipelines), we can put into the column transformer that we want to scale the numerical features and encode the categorical features, and it does that when we fit and transform the data.

4. A Randomized search is similar to a grid search, but much more efficient as instead of extensively going through all the parameter grid we choose, it randomly chooses without replacement n combinations of parameters (n = 10 by default). Image below taken from the article from Bergstra and Bengio, 2012, link on the references.



5. The Extremely Randomized Trees Classifier is a very similar algorithm to the Random Forest Classifier, which fits randomized decision trees – it randomizes the node split instead of looking at the best thresholds. Because of this, it usually has less variance and overfitting. It may be better or worse than the Random Forest, depending on the dataset.

6. Voting Classifier is an ensemble model which uses base estimators' predictions of the target label to then combine them in two possible ways: "hard voting" – the majority label in each row for every model is chosen as the final prediction, or "soft

voting” – the final label is computed using the average of all predicted probabilities of the belonging to each class.

7. Cross_val_predict allows us to see the predictions for validation folds in a cross-validation method.