HOSPITAL READMISSIONS

Group 8:

20230428	Flávia Rodrigues
r20201580	Inês Silva
20230464	Marta Jesus
20230779	Nicolas Zerene
20230477	Tomás Araujo

INDEX

ABSTRACT

This report aims to explain the process of data pre-processing and model selection according to specific datasets. The objective is to investigate the likelihood of individuals returning to a hospital along with the frequency and time span of their return visits. For instance, the analysis will assess whether they return within less than 30 days or not.

Consequently, it shall provide the tools to intervene in this issue, as a result, give effective approaches to tackle rehospitalisation rates. These can help with the shortage of both financial and human capital.

Therefore, the first step was to understand the available data and transform it, into becoming useful and accurate for the project. This includes data cleaning (addressing Nan values and outliers), performing feature selection and creation (involving dropping unusable variables and creating new ones) as well as formatting the data to be compatible with the models. To make an informed decision many models were created, trained, and tested according to their accuracy, precision, recall and f1 scores. The selection process aimed to identify the best-performing model, which would then be applied to the validation data.

To sum up, the results of this research paper led to optimising the hospital's performance.

I. INTRODUCTION

Nowadays, hospitals are overwhelmed with challenges and deficiencies, especially in serving the less privileged. Sometimes these people have to delay going to the hospital for as much as they can, aggravating their health conditions as well as increasing the probability of recurrent visits. No matter the reason why someone goes to the hospital, the reality is the more people that return or go to the hospital, the higher the workload is. This situation contributes to inefficiency, overcapacity, and financial burden. Challenges, all of which, negatively impact the life of patients.

Thus, predicting the likelihood of someone recurring in the hospital for further care can help improve the hospital's quality of services as they have more insights into the problem. In retrospect, by knowing the causing factors, more accurate solutions can be found. By using cutting-edge machine learning techniques, this study shall explore the patient's characteristics and needs, namely, the medical records, to understand what hospitals are lacking to change the current reality. To reach the best results, it was of the utmost prominence to do a thorough preprocessing phase of the data together with scaling it. This approach ensures finding the most appropriate model to achieve the goal.

Our training data had 71236 observations and 31 features, including 2 targets variables. This dataset will be utilised to train the models while the test dataset, with 30530 observations, will evaluate the model performance on unseen data.

II. DATA EXPLORATION AND PREPROCESSING

Composed of a wide range of steps, these stages of the study hold significant importance to the quality of the solution being explored. It is during this phase that the greatest insights about the patients are uncovered. Leading to innovative perspectives that not only enable feature selection but also feature engineering.

Claiming to use the available methods in Python, first many packages were imported, to be more precise, *pandas, numpy, random, matplot, seaborn, scipy, time, warnings, imblearn and sklearn*. Although with different purposes, all played a crucial role in the results obtained. Next, the csv documents, *Train,* and *Test*, were imported using the pandas *read_csv* method.

Moving on, the partition of the datasets was made to facilitate the selection of an appropriate model for the given data. It is fundamental to split the data before processing it to prevent data leakage, since it can happen if not appropriately divided. To do this, the targets for each prediction were defined, being '*reamitted_binary*' and '*readmitted_multiclass*' for the binary and multiclass predictions respectively. Also, these variables were dropped from the training data set as they could not be part of the features for the predictive models. Hence, the train dataset was divided randomly into two parts: a new train dataset (80% of the previous data) to the model learning the underlying patterns. Consequently, this is why the new validation dataset (20% of the data) is vital to allow comparison of model performance as well as to avoid overfitting them into the training data. This division was performed in distinctive sizes (70%/30%) but the one initiated was the one with the best results.

Understanding the data is essential to a good performance so, naturally, the next step was checking whether a variable was numerical or categorical. It would be classified as numerical if the datatype of the column was *int64* or *float64*, whereas the remaining were categorical (*object* datatype).

There are multiple possible approaches to deal with outliers. In this project two options were considered, manual and IQR methods. Starting with the simpler way, the numerical features were plotted into box plots which showed that many of them presented quite substantial incoherencies (fig. 1). Formerly, by observing feature by feature and deciding where the problems were, the function *outliers_to_nan* allowed to change all the outliers in the different datasets into NaN values. This function would run the columns specified in *columns_and_thresholds* dictionary, checking if any value would be superior to its threshold (fig. 2). If they were, it would be considered an outlier, requiring it to be changed. To make them coherent values, the mode function was applied. By doing this, only 1.36% of the data was considered outlier and changed. The IQR method could also be useful to remove outliers automatically, however the percentage of kept data went to 64.95 so a lot of observations would be changed, and it would not be viable for the project. For the categorical features, the count of the values was checked then plotted into histograms so the values could be better understood. No outliers were considered so these features will be handled later (fig. 3).

To evade treating irrelevant data, prevent the model from learning from repeated instances, the next step was checking for duplicates. In this case, even though there were no duplicates, the *drop_duplicates* function was applied in all distinctive copies of datasets, such as train, validation, and testing. These copies replaced the original datasets to allow making changes without affecting them and avoid having to run the entire notebook every time there is an error or when the code is improved.

Missing values can present themselves in diverse ways including change in accordance with the data available. In this situation, it was highlighted the fact that a great deal of the variables had a high percentage of NaN values, that were checked with the function *isnan*. Moreover, when taking a closer look into the data some variables presented hidden missing values, in particular, *'?', '', 'Unknown/Invalid'*, when classified into their columns, hold no meaning. Equally it reduces bias and inaccurate results as well as some machine learning algorithms do not support missing values, this step is essential.

While the variables *glucose_test_result* and *a1c_test_re*sult have a high percentage of missing values, they could be significantly important for the good performance of the model as they are part of routine diabetes exams. Taking this into account, instead of dropping them, the NaN values were swapped to "No test done" through the *replace* method. Posteriorly, these features were transformed into a new variable, *Exams_performed_results*, that followed the conditions: if both had the value "No test done", it returned this same result, else it would concatenate together *glucose_test_result*: plus its value, *a1c_test_result*: plus its value. Followed by dropping the original variables.

The missing values in *payer_code* were replaced by "No health insurance used" given the fact an assumption was made, if someone does not present insurance when checking in, they probably do not have it. This will later influence the project to transform this variable into a new binary one, *Insurance_or_not*, indicating whether someone used insurance instead of the type of insurance implemented.

Initially, the missing values of the variables *race, gender, age, admission_type, discharge_disposition, admission_source,* and *primary_diagnosis* were solved by replacing the NaN values with the variable's mode, but other changes will be conducted further on the project to improve the quality of the data.

Subsequently, it was decided that changes were needed to the *primary_diagnosis*, *secondary_diagnosis* and *additional_diagnosis*. If the *additional_diagnosis* had missing values*,* these were filled with '000'. For the *secondary_diagnosis,* something new was tried when the values from *additional_diagnosis* when available, if these values were still missing, they were filled with '000'. For *primary_diagnosis* the same logic as before was applied, embracing the values of *secondary_diagnosis* when needed and '000' became the alternative. Any remaining missing values in each column are filled with '000', leaving no gap behind (fig. 4).

Successively, variables that did not add value were dropped, specifically, *country, weight* and *medical_specialty.* Their removal is explained by their limited variability, as the country contained only one value while *weight* and *medical_specialty* had 96.8% and 49.1% of missing values, respectively.

The features *change_in_meds_during_hospitalization, prescribed_diabetes_meds*, and *gender* only assumed two values each, thus they were transformed into binary variables which either undertook 0 or 1 values. Additionally, for simplification purposes, *age* was changed to values between 0 and 9 according to specific intervals. Regarding *discharge_disposition*, went through a simplification process reducing the twenty-five original categories to only four new ones: “Going home”, “Going to a medical facility”, “Lost track” or “Stays in hospital”, creating a new variable called *discharge_category.*

Furthermore, new variables were created such as *total_procedures* which is the sum between the number of lab and non-lab tests, allowing to drop of the original variables*,* *number_lab_tests* and *non_lab_procedures.*

Medication was changed whenever an individual did not specify any medications’ names, becoming “No medication used”. Complementing it, the values were changed to a string type in order to create a single value making it easier to compare. Later on, a top 10 of the most applied medications was created and if any of the values was not one of these, it would be categorised as “Other Combination”. This new modification led to create the variable *new_medications* which made the variable *medication* outdated, ready to be dropped.

Outside of the scope of the variables given it was made the experiment of categorising the diseases through their ICD9 code against an external list found and applying it to the primary, secondary and additional diagnosis columns. These were evaluated into colours by the gravity of the disease being green (1 point), yellow (2 points), and red (3 points), increasing gradually according to the colour (0 points represents no diagnosis appointed). Next, along with these risk points, *age points* were created since the risk increases with age. Consequently, the feature *total_points* was created through the sum of disease and *age points*, making the original variables unnecessary and eliminated.

Poorly organised, *admission_type* and *admission_source* were composed of a huge variety of categories which made it close to unreadable. Therefore, in *admission_type* if a value was either of these, “Not Available”, “Not Mapped”, “Trauma Centre”, or “Newborn”, it became “Not Tracked”. While, in *admission_source*, if the value was not part of the most frequent admission sources, being the threshold 1000, then it would be classified as “Other”, allowing an easier interpretation of the data.

Further into research, it was observed that a different variable could raise, *total_visits* which includes both *inpatient_visits_in_previous_year, outpatient_visits_in_previous_year*, and *emergency_visits_in_previous_year*. This way gives the full picture and instead of having three variables related to the same problem, there was a unique one, adding all the three.

What is more, the feature *medications_per_day* was created by performing a division between the *number_of_meds* prescribed and the *length_of_stay_in_hospital,* this way was counted how many medications a patient took during the encounter.

Another candidate for a variable was *Number_encounter* which is a group by of encounters by *patient_id* which allows one to see how many times the patient attended the hospital, making both variables engaged unrequired to proceed with the project, finalising, with being dropped.

In conclusion of this step, all variable types were checked and some (gender, change_in_meds_during_hospitalization, prescribed_diabetes_meds, Insurance_or_not) were transformed into a *boolean* data type, owning to the fact these only took 0’s and 1’s making them more suitable for analysis.

On the execution of the above-mentioned steps, concerns might be raised. For this reason, feature selection was a tool to mitigate the duration of training and overfitting, along with enhancing accuracy by selecting the most relevant features. As features were changed both non-metric and metric were as well (fig. 5).

Using Stratified K-Fold, it was possible to randomly select ten different samples without repeating any of the records, allowing to decrease biasness along with variance, simultaneously, strengthening generalisation and performance.

The categorical variables were the ones handled first applying the chi-squared which is an independence test between each categorical feature and the target feature, it provided a sum up table for each variable showing whether a variable should be discarded or kept. Afterwards, the target feature assumed either the 0 or 1 values (fig. 6). Following up, it was necessary to encode the non-metric variables using a *target encoder*, encoding each category according to the closest average value of the target feature. To complement this decision-making process, it was also performed the mutual information method to measure the relationship of these variables with the target one. The results reinforced the decision to drop *admission_source* and *new_medications* since the score was lower than 0.0045 (this value was decided the last time the notebook was run due to the fact mutual information provides random values that can change every time) (fig. 7).

Many times, some features dominate others and to improve performance it was indispensable to standardise the data as it puts all data into the same scale, to have equal importance and to allow well-based comparisons. The technique that showed to be superior was MinMaxScaler, proven effective due to its simplicity and effectiveness in preserving relationships between features, it puts the values between a range of 0 and 1. Although sensible to outliers, many of those were already dealt which should not cause major problems.

For the numerical features, the variance of the variables can provide a great deal of information about them. So, it was calculated to see how suitable a feature was and if the variance is 0 (univariate) then it should be dropped, yet in this project it did not lead to any changes. Furthermore, to analyse the relationships between the variables, the Spearman correlation was checked on account that this approach seemed more suitable compared with Pearson as the data being cast off is ordinal data. This technique can be useful since if a variable has a high correlation with another (>0.8) it probably means one of them is not as valuable as thought (fig. 8). The only pair with a high correlation was *age* and *total_points* considering that this last feature was created using *age*. Using the Wrapper methods, Recursive Feature Elimination (RFE) using Logistic Regression and Decision Tree as an estimator, evaluating the importance of each variable, and returning the optimal number to keep. The number of selected features was defined with a minimum of four regarding having the best average score possible (fig. 9). However, it was not enough to decide so Embebed methods were also executed. Lasso eliminates features that do not seem to have importance by giving them a coefficient of zero. However, in this case, no feature was eliminated by Lasso (fig. 10), but only nine out of twelve were selected in all the folds, so they were the ones chosen. In the Ridge method only two variables were selected across all folds (fig. 11). For the Decision Tree one, the Gini Coefficient, as well as the Entropy, were visualised and helped understand which features should be kept, considering the ones with a value greater than 0.05 (fig. 12). Last but not least, ANOVA was done as it checks the means of the features that are significantly different from diverse groups (fig. 13).

In conclusion, this decision ended with dropping the features *gender, average_pulse_bpm, change_in_meds_during_hospitalization, prescribed_diabetes_meds,* and *'Insurance_or_not',* on the grounds these all had more than three discards over all the methods (fig. 14). By the end of preprocessing, the dataset had 11 features.

III. BINARY CLASSIFICATION

Developing the binary classification models required treating the sample. Therefore, being the dataset imbalanced, both under and over sampling techniques were tested so that it would be possible to find the most accurate prediction. For this project, the goal was not only to look for the best F1 score but also to look for other classification metrics such as precision, to show how accurate the positive predictions are this is the true positives among all positives, and recall that check if the model identifies the positive instances correctly.

The first thing to be done was to verify if the data was balanced by examining the value counts of the target variable. The result was a proportion of, approximately, 0.88 to 0.11, confirming the imbalance within the dataset. To contour this the over sampling technique, SMOTE, was applied as it tries balancing completing the minority class by duplicating records so that the size is equal in both, reducing biasness, this way simplifying the model to avoid overfitting and improving performance.

It was essential to make a model selection where the data was tested by using the following models, *Logistic Regression*, *KNeighboursClassifier*, *MLPClassifier*, *DecisionTreeClassifier* and *GaussianNB*. In the end, the results displayed the MLPClassifier as the best which made the study inclined to select the Neural Networks Model (fig. 15). Nevertheless, there was still room for improvement so the best hyperparameter where tested and selected for each model.

Starting with the perfection of the KNN model, first, it was explored how many neighbours were required to obtain a better result, leading to simply using two neighbours. Nonetheless, other parameters should be explored, namely, weights – uniform or distance, p (decides the type of distance to be utilised, Manhattan or Euclidean) – 1 or 2, and algorithm – auto, ball_tree, kd_tree or brute. Being the best parameters **'algorithm': 'auto', 'n_neighbours': 2, 'p': 1, 'weights': 'distance'** after using GridSearchCV to reach them (fig. 16).

The Logistic Regression has the parameters solver, penalty, and C so once again RandomizedSearchCV was applied, reaching the result **'solver': 'saga', 'penalty': 'l1', and 'C': 10.0,** which are the parameters that pointed to be the best (fig. 17).

Taking into consideration that the Decision Tree model takes a lot of factors and doing it through GridSearchCV would take too long and overload the notebook. Therefore, each parameter was separately evaluated which indicated using **'max_depth': 1, 'min_samples_leaf': 188, 'min_samples_split': 23,** without defining splitter and criteria as it would decrease the f1 score (fig. 18).

Neural Networks was a little more complex to explore so a RandomizedSearchCV was run to decide all the hyperparameters. The best parameters indicated were **'activation': 'relu', 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.015, 'solver': 'adam'** (fig. 19).

Naïve Bayes only had one parameter being explored so the overload would be smaller, once again it was applied GridSearchCV to 'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5] that gave the result **'var_smoothing': 1e-09** (fig. 20).

To sum up, the F1 score applied to both train and validation, overfitting, recall and precision score, along with the ROC curve (gives a representation of the compromise between the True Positive Rate and the False Positive Rate), the models Neural Networks and Logistic Regression showed the most promising results (fig. 21,22). As the data is imbalanced, focusing on recall and precision was the approach taken to make a well-based decision on the best model. After the threshold plot, Logistic Regression emerged to be the one with the best result for the over sampled data (fig. 23).

For the under sampling, using RandomUnderSampler, these balances both minority and majority classes by decreasing the size of the majority class. Although different, both can improve results and avoid overfitting.

As previously done, it was important to make model selection, showing that Neural Networks was the most promissory (fig. 24). However, these models did not show any encouraging results, so they needed to be optimised. As under sampling reduces the size of the data, the processing time was lower so different techniques were employed.

Applying the same approach as before it was concluded that KNN's best parameters are: **'algorithm': 'auto', 'n_neighbours': 2, 'p': 2, 'weights': 'distance'** (fig. 25). The Logistic Regression was executed with GridSearchCV, the best parameters were **'C': 0.01, 'penalty': 'l2', 'solver': 'sag'** (fig. 26). Following a similar approach as in over sampling, the best parameters for the Decision Tree model were **'max_depth': 5, 'min_samples_leaf': 82, and 'min_samples_split': 178** (fig. 27). Moreover, like what happen in over sampling some parameters would jeopardise the quality of the model, so they were not included. Neural Networks was a little more complex to explore, as well as Decision Trees would take too much time and computer power, so its constraints were searched separately. After this, a GridSearchCV was run to decide the hidden layer's sizes as this was the only parameter not tested individually. The best parameters indicated were **'activation': 'relu', 'hidden_layer_sizes': 8, 'learning_rate': 'constant', 'learning_rate_init': 0.034, 'solver': 'lbfgs'** (fig. 28). Naïve Bayes only had one parameter being explored so once again after the usage of the GridSearchCV, the result was **'var_smoothing': 1e-09** (fig. 29).

In brief, after checking the F1 score for both train and validation, overfitting, recall and precision scores along with the ROC curve, models Neural Networks and Decision Trees were the ones with the best results (fig. 30, 31). Therefore, by once again considering recall and precision to make a more accurate decision about this imbalanced data, a threshold was plotted based on these metrics, Decision Trees was the one with the best result for under sampling data (fig. 32).

Hopping the results would improve, ensemble methods were applied. Ensembled models investigate optimising the collective wisdom of multiple models to increase the accuracy of the predictions. The methods Bagging, Random Forest, Gradient Boost, Stacking Classifier and AdaBoost Classifier were executed to produce this leverage and applied to under sampling because the computational time for the over sampling was too elevated to get results. It was also necessary to apply once again the *Stratified K-Fold* strategy.

For this were created two functions *return_f1score* and *return_results* according to the f1 score to get the best parameters for each model.

Bagging was tested with the combination of Decision Trees and KNN with the Bagging Classifier, the result showed that the best model is Decision Trees with Bagging. It was required to test different values for the different parameters to create an optimal model, **'n_estimators': 100, 'max_samples': 0.2, 'bootstrap': True,** and **'bootstrap_features': True,** was the best result (fig. 32). For Random Forest was tested, leading to **'n_estimators': 300, 'bootstrap': True, 'max_samples': 0.2, 'max_dept': 8** (fig. 33). Gradient Boost after explored pointed to **'learning_rate': 0.1, 'n_estimators': 150, 'subsample': 0.6, 'max_features': None** (fig. 34). For the stacking ensemble, all the previous models were taken into consideration to find the best estimators (fig. 35). The ones with the best results were tested in groups, although none of them got a better result than the Gradient Boost Classifier. The final estimator was Logistic Regression as this is the least complex one to test. Moreover, the Ada Boost estimator considered was the Decision Tree and the **'max_dept'** was tested giving the result of two (fig. 36). To see which one performed the best, it was also tested the Logistic regression as an estimator, the results confirmed that the best one was the Decision Tree, **'n_estimators': 10, 'learning_rate': 0.1,** separately. To see if the learning rate would change based on the estimators, this was tested, and the results showed that having a smaller number of estimators could demand a higher learning rate as this one increased to 0.9. For the final model, the algorithm SAMME was considered.

In brief, for these models were also performed the ROC Curve, all of them got very close results, but the best ones were Random Forest, Gradient Boost and Stacking (fig. 37). Considering recall and precision (fig. 38 and fig. 39), Random Forest had the best result even though the one with the best performance in Kaggle was AdaBoost.

IV. MULTICLASS CLASSIFICATION

This section of the project mostly focused on replicating all the previous steps made for the binary classification and changing it according to the new challenge at hand, which is adding the category of someone not going back to the hospital.

The train-validation split was done considering this new target variable. Pre-processing in this part of the project did not have major alterations as the steps that compose it were carefully designed since the beginning so they would fit in both parts of the project.

Different from the binary classification, the encoding was done by hand, taking the values 0 (Not coming back), 1 (less than 30 days) and 2 (over 30 days) for this target (fig. 40).

The feature selection was redone cause the best features for the binary could not be the same for the multiclass target. For the categorical features, chi-squared test and the mutual information (fig. 41) were done giving the information that both *admission_type* and *race* should be discarded for this model, having different results from the binary one. Considering the numerical features, once again, correlation matrix, recursive feature elimination with both Logistic Regression and Decision Trees, Lasso, Ridge, Decision Trees and ANOVA were tested. After this, twelve features were selected (fig. 42), *gender*, *change_in_meds_during_hospitalization, prescribed_diabetes_meds* and *Insurance_or_not* these were dropped.

However, for the model hyperparameters selection, repeating the full process requires an enormous amount of time and computer power. Although searching for them would be the most appropriate approach, to avoid the problems previously mentioned, the models were applied with the same parameters obtained in the binary models.

Once again, the models experimented on were Logistic Regression, K Nearest Neighbours, Decision Trees, Neural Networks and Naïve Bayes with the metric *average = weighted*. When using over sampling the model with the greatest performance was Neural Networks as it got the least overfitting and decent precision and recall values as well as a great f1 score in validation (fig. 43). For the under sampling, although Decision Trees have slightly negative overfitting, is the one with the overall score (fig. 44).

Nevertheless, ensembled methods were also tested seeking better scores. Stacking, Gradient Boosting and Random Forest got very close validation f1 scores but when compared with the train, the model with the best performance was Random Forest (fig. 45).

V. CONCLUSION

Something is only as good as how much effort you put into it, that is exactly what this project was all about. Data is not always balanced or clean and in this case was neither of those. To reach the results, it required a lot of trial and error, whether changing a variable or accepting it or the parameters that compose the models.

The initial expectations for this project were to help, with all the metrics learned in class, the healthcare system by providing them a model with an optimal performance that would help, at some point, save millions of dollars. Yet, it was challenging due to the fact that our data was imbalanced, and the techniques used to deal with it could lead to incorrect assumptions and overfitting. However, it allowed to get a step closer to a more beneficial approach.

Throughout the project, some aspects could have been handled differently. Nevertheless, as highlighted in the report, constraints such as computational power and time limitations made achieving perfection difficult. The hyperparameters for the models would have been better selected with Grid Search as it is the optimal method to find the best parameters. Nonetheless, the time it requires is extremely high, especially for the over sampled data, which prompted a combination of individual tests and Randomized Search usage instead. Despite this, using the same parameters for both binary and multiclass classification, positive results were obtained for both.

In conclusion, for the binary classification, the model that got the highest score on Kaggle challenge was Decision Trees for over sampled data. Yet, when the ROC curve was performed it did not show great results. So, the final model chosen for this project was Random Forest, showing that around 32% of the individuals will be back before 30 days. While this model did not achieve the highest performance on the Kaggle challenge (0.303) when calculated the optimal threshold it did present the best F1 score (0.281) along with precision and recall being (0.487).

On the other hand, for the multiclass classification there were three models with very close F1 scores on the validation dataset, Random Forest with 0.5393, Gradient Boosting got 0.5425 and Stacking with 0.5421. Having that, the train scores were also calculated, and negative overfitting was found in both Gradient Boosting and Stacking. So, the final model for multiclass classification chosen was Random Forest, to predict whether someone will not be back, or if it will take less than 30 days or more than that.

REFERENCES

Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018b). Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. *Departmental Technical Reports (CS)*.

https://scholarworks.utep.edu/cs_techrep/1209/

Tamboli, N. (2021, October 29). *Tackling Missing Value in Dataset*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

Health, M. of. (n.d.). *Diagnostic Code Descriptions (ICD-9) - Province of British Columbia*. Www2.Gov.bc.ca. https://www2.gov.bc.ca/gov/content/health/practitioner-professional-resources/msp/physicians/diagnostic-code-descriptions-icd-9

Sikaris, K. (2009). The Correlation of Hemoglobin A1c to Blood Glucose. *Journal of Diabetes Science and Technology*, *3*(3), 429–438. https://doi.org/10.1177/193229680900300305

sklearn.preprocessing.TargetEncoder. (n.d.). Scikit-Learn. Retrieved December 20, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.TargetEncoder.html

Mutual Information. (n.d.). Kaggle.com. https://www.kaggle.com/code/ryanholbrook/mutual-information

gajawada, sampath kumar. (2019, October 20). *ANOVA for Feature Selection in Machine Learning*. Medium. https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476

Top Performance Metrics in Machine Learning: A Comprehensive Guide. (n.d.). Www.v7labs.com. https://www.v7labs.com/blog/performance-metrics-in-machine-learning

Ellis, C. (2023, January 21). *Oversampling vs undersampling for machine learning*. Crunching the Data. https://crunchingthedata.com/oversampling-vs-undersampling/

Handling Categorical Data in Python Tutorial. (n.d.). Www.datacamp.com.

https://www.datacamp.com/tutorial/categorical-data

Sharma, M. (2023, October 3). What is ridge regression? An overview. Intellipaat. https://intellipaat.com/blog/what-is-ridge-regression/

Pfeiffer, A. (2019, May 27). A Deeper Look into Feature Selection. Medium.

https://medium.com/@pfeifferanya/a-deeper-look-into-feature-selection-e05d288fcef9

Velayudham, V. (2020, March 6). ROC Curve, AUC value — Significance of thresholds and what do they really mean on the model…. Analytics Vidhya.

https://medium.com/analytics-vidhya/roc-curve-auc-value-significance-of-thresholds-and-what-do-they-really-mean-on-the-model-723039baf35c

ANNEXES

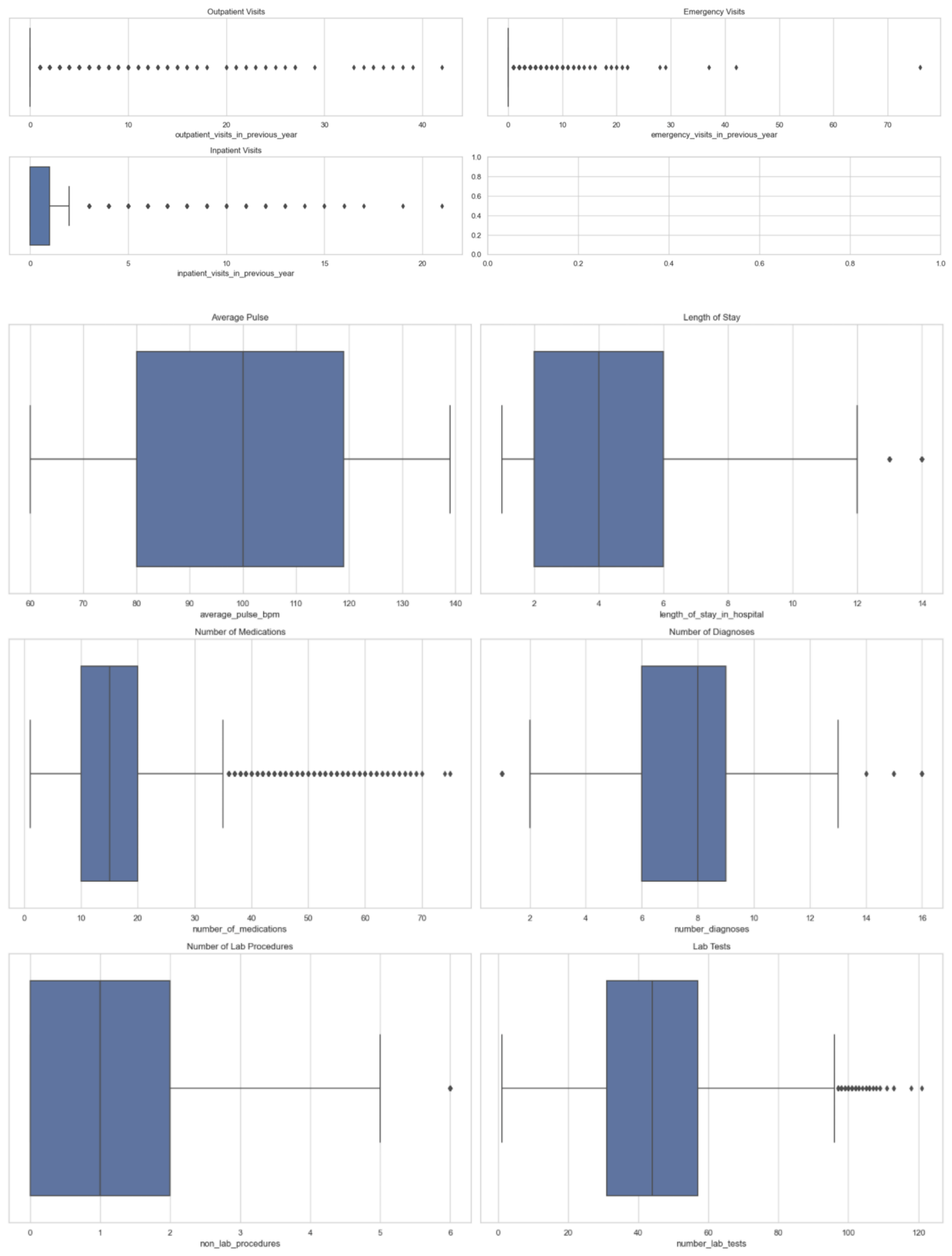
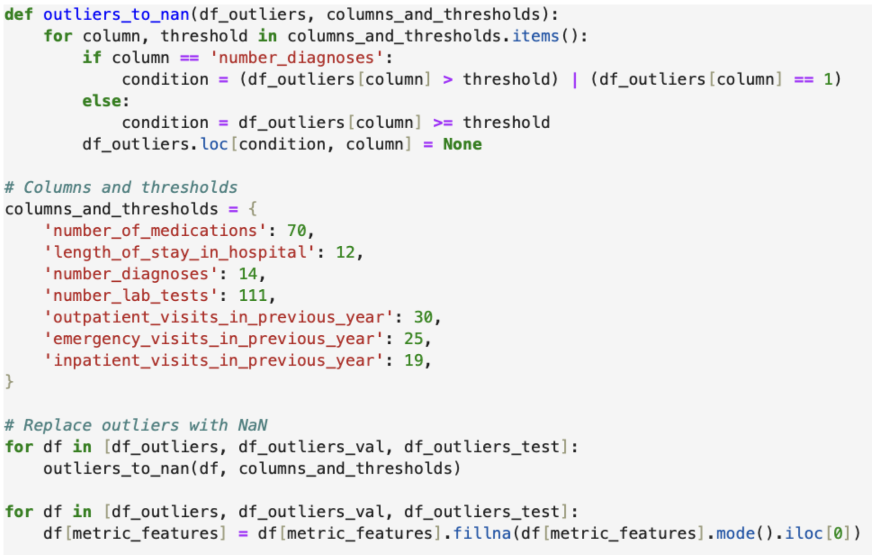


Figure 1- Visualisation of Numerical Variables



```python
def outliers_to_nan(df_outliers, columns_and_thresholds):
    for column, threshold in columns_and_thresholds.items():
        if column == 'number_diagnoses':
            condition = (df_outliers[column] > threshold) | (df_outliers[column] == 1)
        else:
            condition = df_outliers[column] >= threshold
        df_outliers.loc[condition, column] = None

# Columns and thresholds
columns_and_thresholds = {
    'number_of_medications': 70,
    'length_of_stay_in_hospital': 12,
    'number_diagnoses': 14,
    'number_lab_tests': 111,
    'outpatient_visits_in_previous_year': 30,
    'emergency_visits_in_previous_year': 25,
    'inpatient_visits_in_previous_year': 19,
}

# Replace outliers with NaN
for df in [df_outliers, df_outliers_val, df_outliers_test]:
    outliers_to_nan(df, columns_and_thresholds)

for df in [df_outliers, df_outliers_val, df_outliers_test]:
    df[metric_features] = df[metric_features].fillna(df[metric_features].mode().iloc[0])
```

Figure 2- outliers_to_nan function code

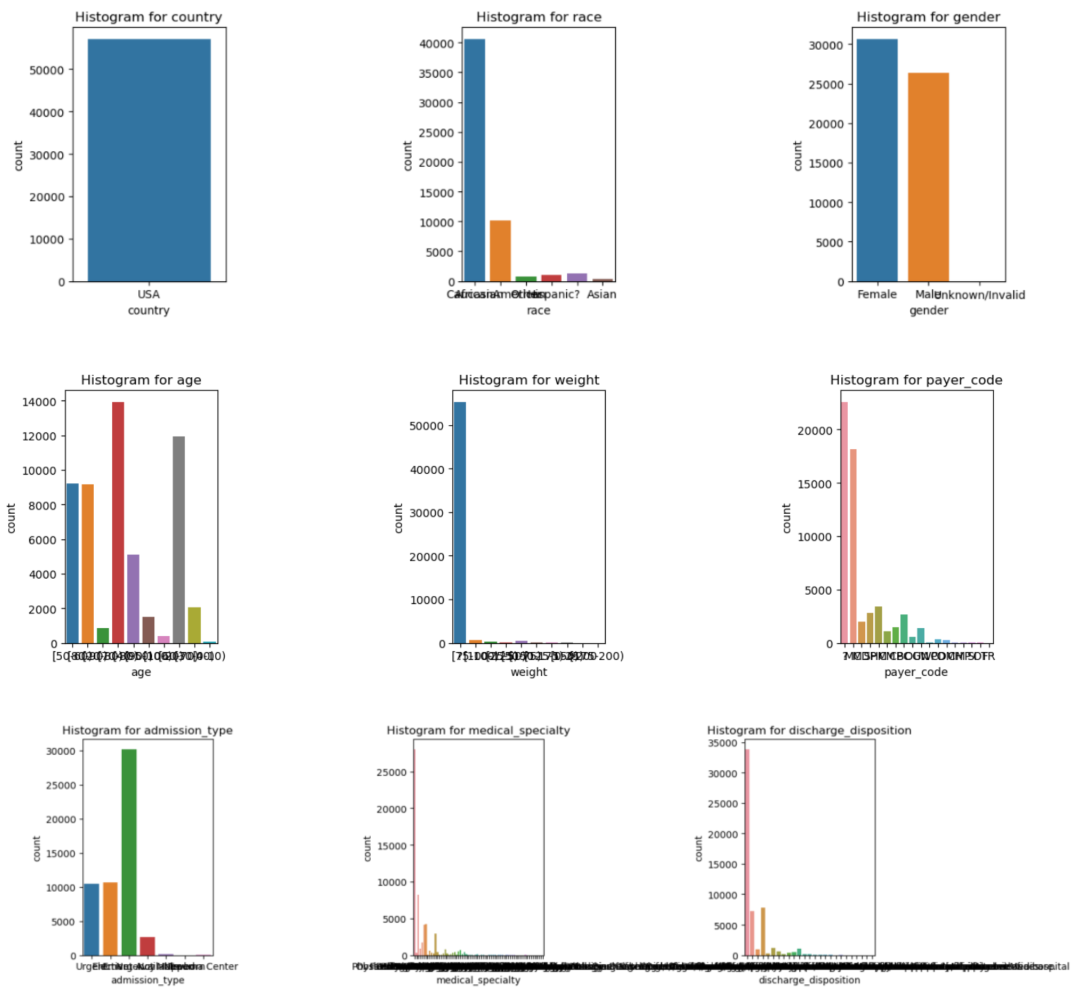
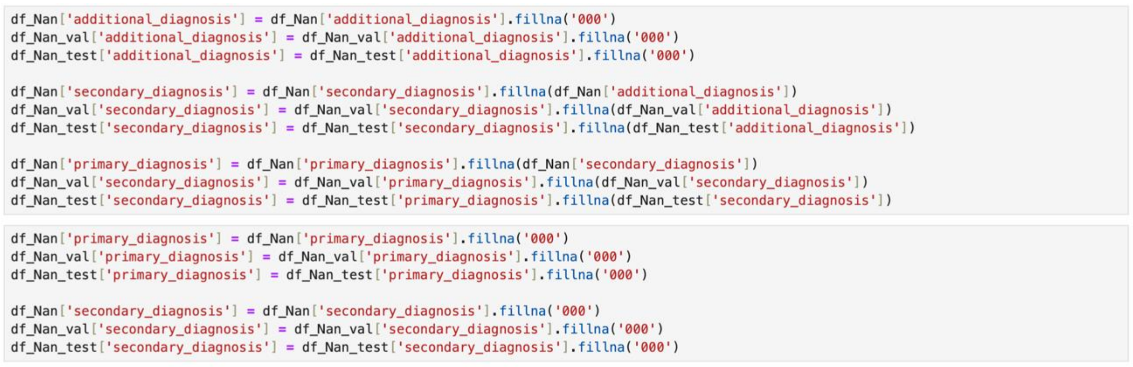


Figure 3- Visualisation of Categorical Variables



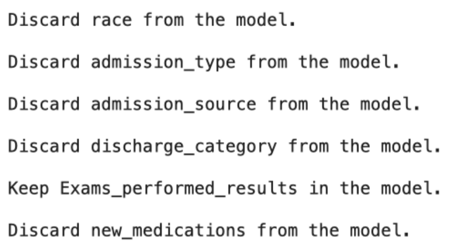
```
df_Nan['additional_diagnosis'] = df_Nan['additional_diagnosis'].fillna('000')
df_Nan_val['additional_diagnosis'] = df_Nan_val['additional_diagnosis'].fillna('000')
df_Nan_test['additional_diagnosis'] = df_Nan_test['additional_diagnosis'].fillna('000')

df_Nan['secondary_diagnosis'] = df_Nan['secondary_diagnosis'].fillna(df_Nan['additional_diagnosis'])
df_Nan_val['secondary_diagnosis'] = df_Nan_val['secondary_diagnosis'].fillna(df_Nan_val['additional_diagnosis'])
df_Nan_test['secondary_diagnosis'] = df_Nan_test['secondary_diagnosis'].fillna(df_Nan_test['additional_diagnosis'])

df_Nan['primary_diagnosis'] = df_Nan['primary_diagnosis'].fillna(df_Nan['secondary_diagnosis'])
df_Nan_val['secondary_diagnosis'] = df_Nan_val['primary_diagnosis'].fillna(df_Nan_val['secondary_diagnosis'])
df_Nan_test['secondary_diagnosis'] = df_Nan_test['primary_diagnosis'].fillna(df_Nan_test['secondary_diagnosis'])
```

```
df_Nan['primary_diagnosis'] = df_Nan['primary_diagnosis'].fillna('000')
df_Nan_val['primary_diagnosis'] = df_Nan_val['primary_diagnosis'].fillna('000')
df_Nan_test['primary_diagnosis'] = df_Nan_test['primary_diagnosis'].fillna('000')

df_Nan['secondary_diagnosis'] = df_Nan['secondary_diagnosis'].fillna('000')
df_Nan_val['secondary_diagnosis'] = df_Nan_val['secondary_diagnosis'].fillna('000')
df_Nan_test['secondary_diagnosis'] = df_Nan_test['secondary_diagnosis'].fillna('000')
```

Figure 4- Code to deal with missing values in primary_diagnosis, secondary_diagnosis and additional_diagnosis

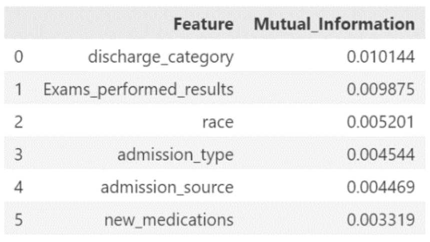
Metric Features		Non-Metric Features
'gender',	'total_procedures',	'race',
'age',	'Insurance_or_not',	'admission_type',
'average_pulse_bpm',	'total_points',	'admission_source',
'number_diagnoses',	'total_visits',	'discharge_category',
'change_in_meds_during_hospitalization',	'medications_per_day',	'Exams_performed_results',
'prescribed_diabetes_meds',	'Number_encounter'	'new_medications'

Figure 5- Categorisation of features into Metric and Non-Metric



```
Discard race from the model.

Discard admission_type from the model.

Discard admission_source from the model.

Discard discharge_category from the model.

Keep Exams_performed_results in the model.

Discard new_medications from the model.
```

Figure 6- Chi-Squared Results



	Feature	Mutual_Information
0	discharge_category	0.010144
1	Exams_performed_results	0.009875
2	race	0.005201
3	admission_type	0.004544
4	admission_source	0.004469
5	new_medications	0.003319

Figure 7- Mutual Information Results

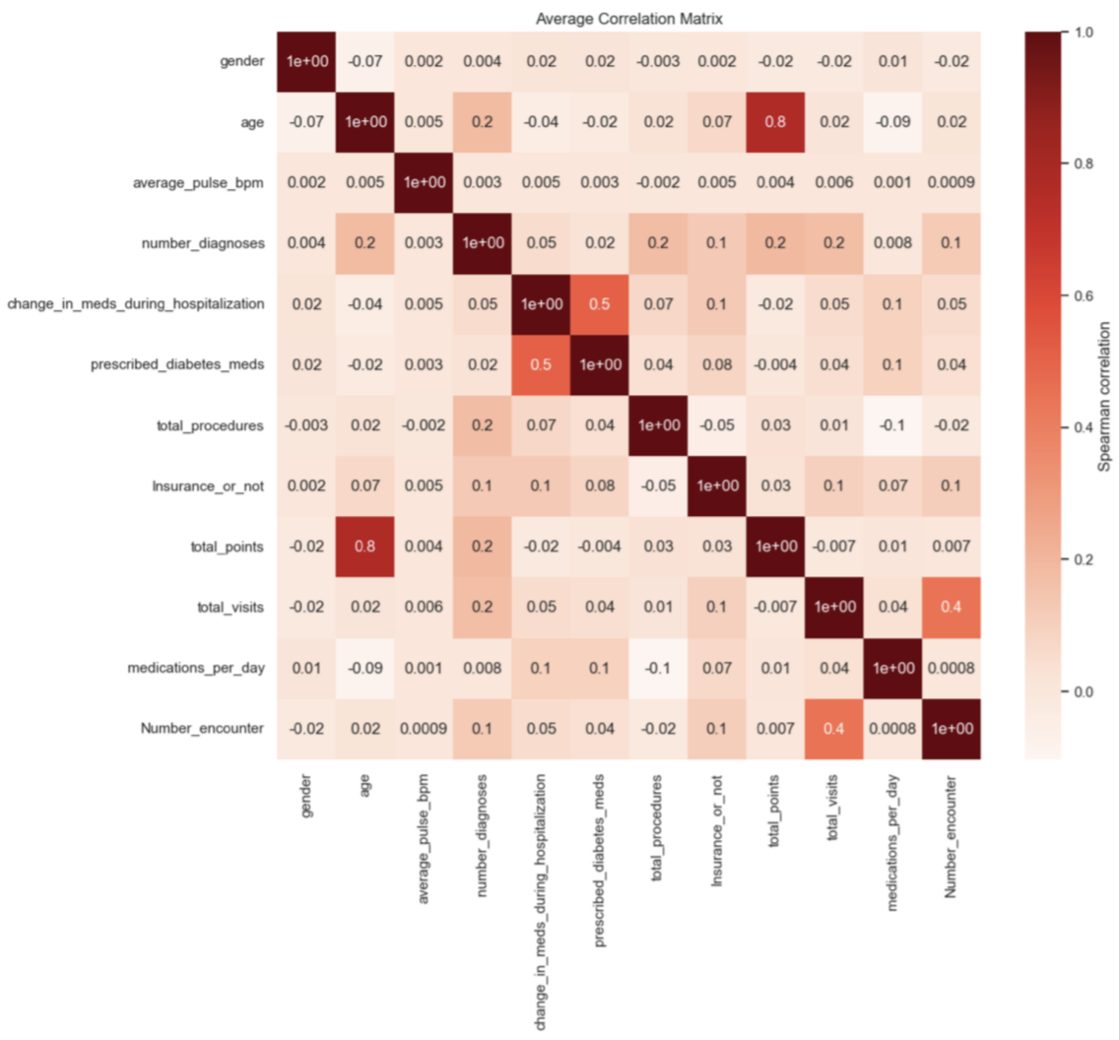
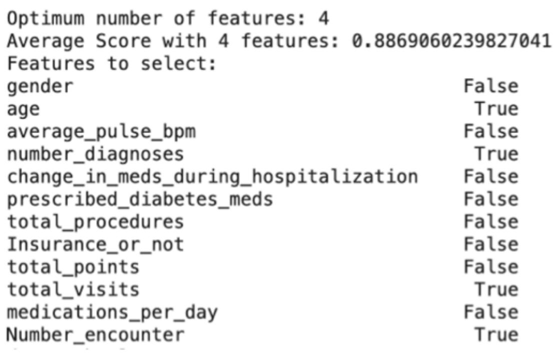


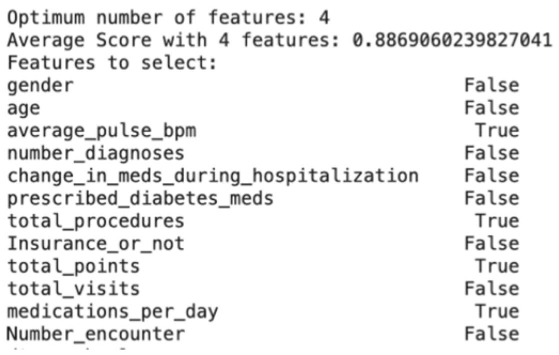
Figure 8- Correlation Matrix

RFE with Logistic Regression



```
Optimum number of features: 4
Average Score with 4 features: 0.8869060239827041
Features to select:
gender                                  False
age                                      True
average_pulse_bpm                       False
number_diagnoses                         True
change_in_meds_during_hospitalization   False
prescribed_diabetes_meds                False
total_procedures                        False
Insurance_or_not                        False
total_points                            False
total_visits                             True
medications_per_day                     False
Number_encounter                         True
```

RFE with Decision Trees



```
Optimum number of features: 4
Average Score with 4 features: 0.8869060239827041
Features to select:
gender                                  False
age                                     False
average_pulse_bpm                        True
number_diagnoses                        False
change_in_meds_during_hospitalization   False
prescribed_diabetes_meds                False
total_procedures                         True
Insurance_or_not                        False
total_points                             True
total_visits                            False
medications_per_day                      True
Number_encounter                        False
```

Figure 9- RFE Results

```
Total features: 12
Selected features across all folds: 9
Variables eliminated by Lasso: 0
Selected features: {'age', 'number_diagnoses', 'Number_encounter', 'change_in_meds_during_hospitalizat
ion', 'medications_per_day', 'total_visits', 'prescribed_diabetes_meds', 'total_procedures', 'Insuranc
e_or_not'}
```

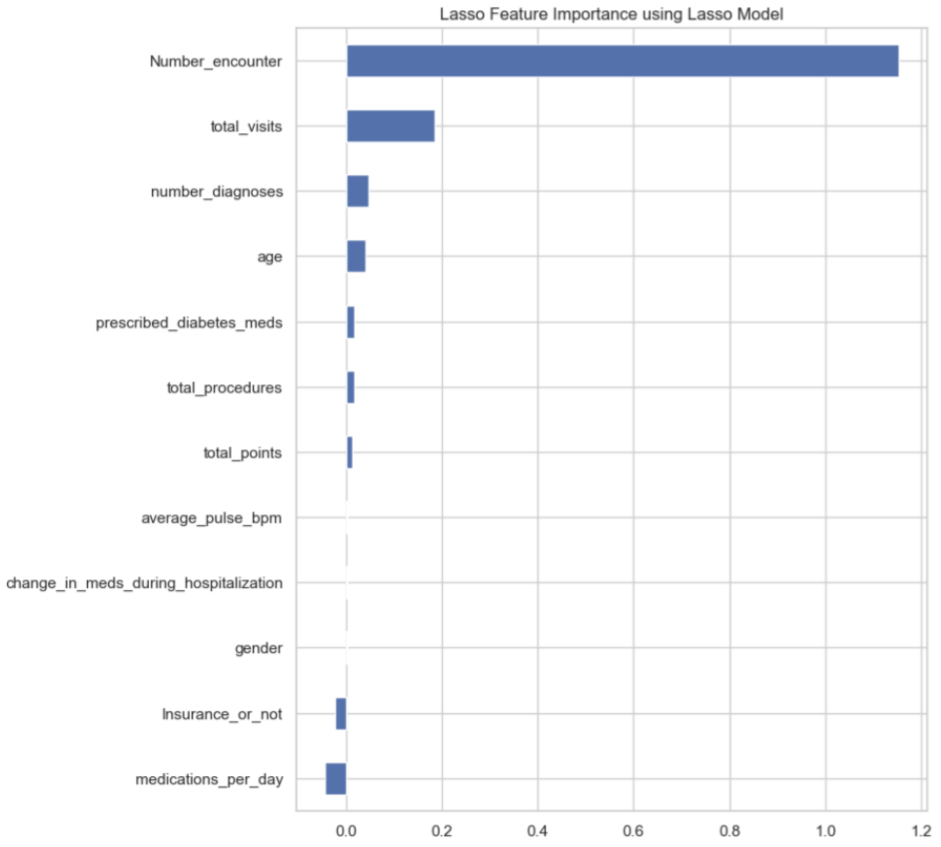


Figure 10- Lasso Results

```
Total features: 12
Selected features across all folds: 2
Features with coefficients shrank to zero: 0
Selected features: {'total_visits', 'Number_encounter'}
```

Figure 11- Ridge Results

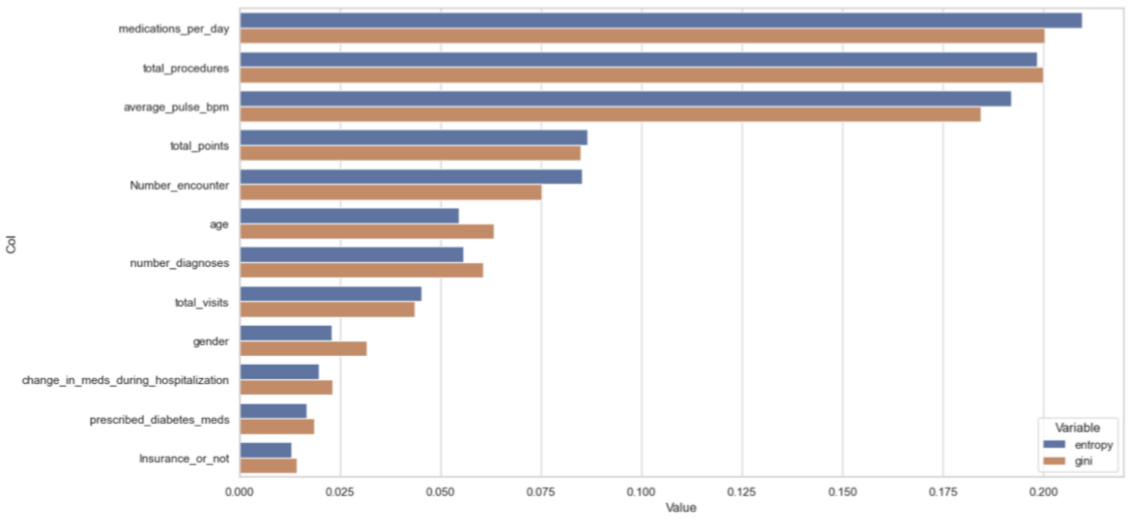
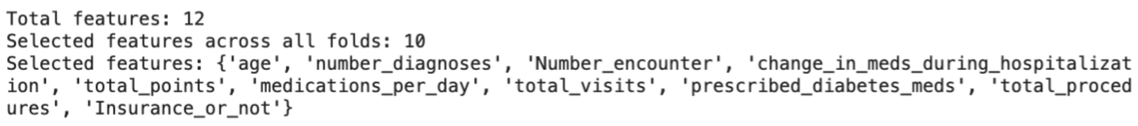
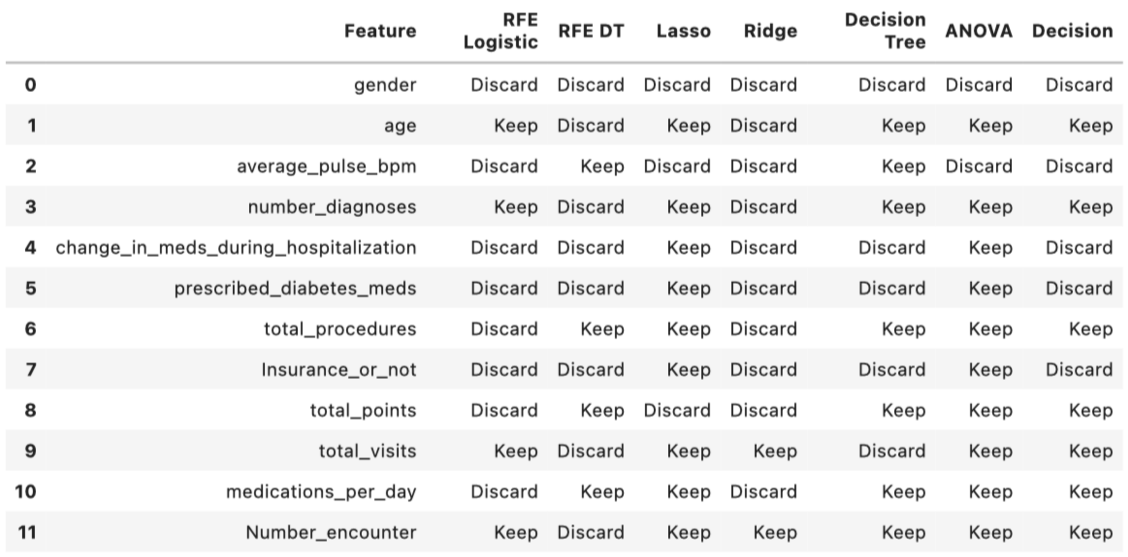


Figure 12- Decision Tree Results



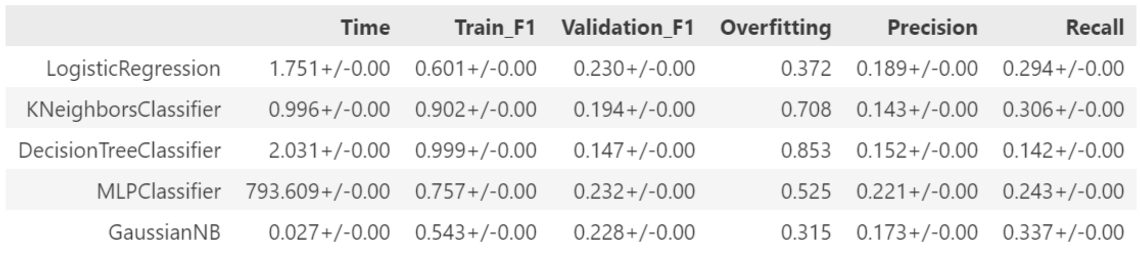
```
Total features: 12
Selected features across all folds: 10
Selected features: {'age', 'number_diagnoses', 'Number_encounter', 'change_in_meds_during_hospitalizat
ion', 'total_points', 'medications_per_day', 'total_visits', 'prescribed_diabetes_meds', 'total_proced
ures', 'Insurance_or_not'}
```

Figure 13- ANOVA Results



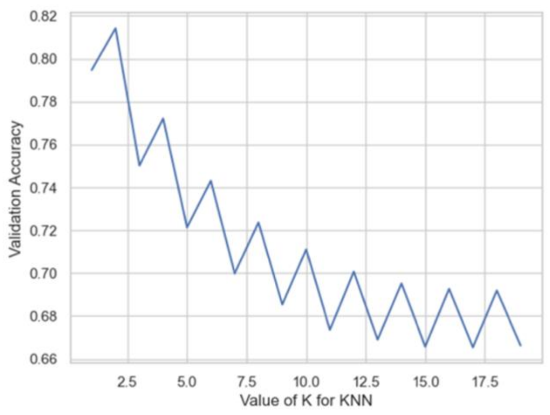
	Feature	RFE Logistic	RFE DT	Lasso	Ridge	Decision Tree	ANOVA	Decision
0	gender	Discard	Discard	Discard	Discard	Discard	Discard	Discard
1	age	Keep	Discard	Keep	Discard	Keep	Keep	Keep
2	average_pulse_bpm	Discard	Keep	Discard	Discard	Keep	Discard	Discard
3	number_diagnoses	Keep	Discard	Keep	Discard	Keep	Keep	Keep
4	change_in_meds_during_hospitalization	Discard	Discard	Keep	Discard	Discard	Keep	Discard
5	prescribed_diabetes_meds	Discard	Discard	Keep	Discard	Discard	Keep	Discard
6	total_procedures	Discard	Keep	Keep	Discard	Keep	Keep	Keep
7	Insurance_or_not	Discard	Discard	Keep	Discard	Discard	Keep	Discard
8	total_points	Discard	Keep	Discard	Discard	Keep	Keep	Keep
9	total_visits	Keep	Discard	Keep	Keep	Discard	Keep	Keep
10	medications_per_day	Discard	Keep	Keep	Discard	Keep	Keep	Keep
11	Number_encounter	Keep	Discard	Keep	Keep	Keep	Keep	Keep

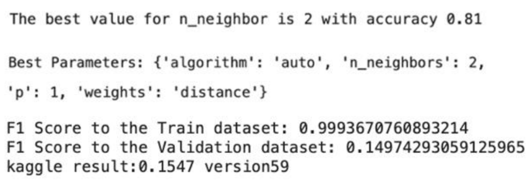
Figure 14- Feature selection sum up



	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	1.751+/-0.00	0.601+/-0.00	0.230+/-0.00	0.372	0.189+/-0.00	0.294+/-0.00
KNeighborsClassifier	0.996+/-0.00	0.902+/-0.00	0.194+/-0.00	0.708	0.143+/-0.00	0.306+/-0.00
DecisionTreeClassifier	2.031+/-0.00	0.999+/-0.00	0.147+/-0.00	0.853	0.152+/-0.00	0.142+/-0.00
MLPClassifier	793.609+/-0.00	0.757+/-0.00	0.232+/-0.00	0.525	0.221+/-0.00	0.243+/-0.00
GaussianNB	0.027+/-0.00	0.543+/-0.00	0.228+/-0.00	0.315	0.173+/-0.00	0.337+/-0.00

Figure 15- General Models Performance with Over Sampling



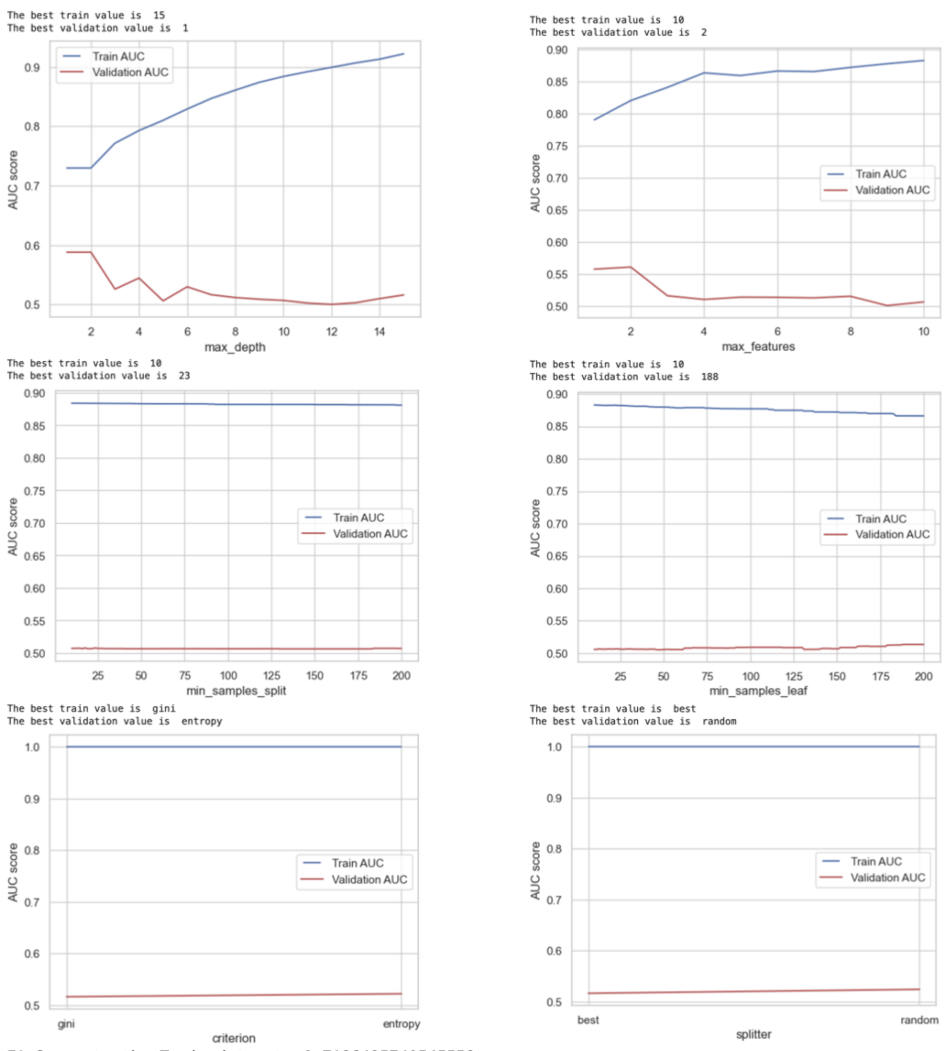


```
The best value for n_neighbor is 2 with accuracy 0.81

Best Parameters: {'algorithm': 'auto', 'n_neighbors': 2,
'p': 1, 'weights': 'distance'}

F1 Score to the Train dataset: 0.9993670760893214
F1 Score to the Validation dataset: 0.14974293059125965
kaggle result:0.1547 version59
```

Figure 16- KNN Over Sampling best parameters

```
Best Parameters: {'solver': 'saga', 'penalty': 'l1', 'C': 10.0}

F1 Score to the Train dataset: 0.5998458984014673
F1 Score to the Validation dataset: 0.2284880894506563
kaggle result: 0.266 version 60
```

Figure 17- Logistic Regression Over Sampling best parameters



```
The best train value is  15
The best validation value is  1

The best train value is  10
The best validation value is  2

The best train value is  10
The best validation value is  23

The best train value is  10
The best validation value is  188

The best train value is  gini
The best validation value is  entropy

The best train value is  best
The best validation value is  random
```

```
F1 Score to the Train dataset: 0.7196495740545559
F1 Score to the Validation dataset: 0.25962348974431015
kaggle result: 0.3104 version 61
```

Figure 18- Decision Tree Over Sampling best parameters

```
Best Parameters: {'solver': 'adam', 'learning_rate_init': 0.015, 'learning_rate': 'adaptive', 'hidden_layer_sizes': (50,), 'activation': 'rel
u'}

F1 Score to the Train dataset: 0.7600904221890799
F1 Score to the Validation dataset: 0.2402562733582488
kaggle result:0.2896 version 36
```

Figure 19- Neural Networks Over Sampling best parameters

```
Best Parameters: {'var_smoothing': 1e-09}

F1 Score to the Train dataset: 0.5422730422730423
F1 Score to the Validation dataset: 0.22676822633297064
kaggle result: 0.2505 version 63
```

Figure 20 - Naïve Bayes Over Sampling best parameters

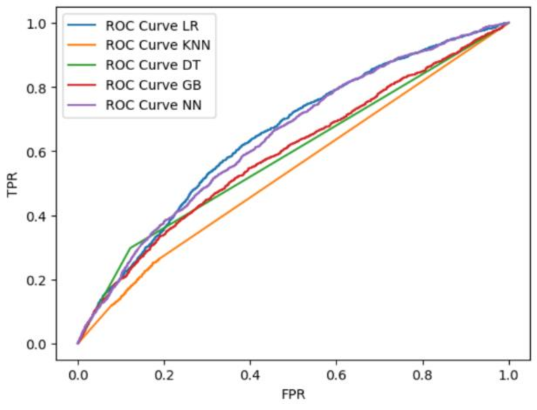
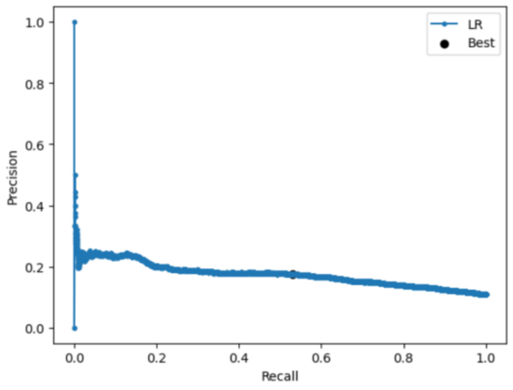


Figure 21- ROC curve for over sampling

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	1.745+/-0.00	0.603+/-0.00	0.231+/-0.00	0.372	0.185+/-0.00	0.308+/-0.00
KNeighborsClassifier	0.548+/-0.00	0.999+/-0.00	0.150+/-0.00	0.850	0.149+/-0.00	0.150+/-0.00
DecisionTreeClassifier	0.257+/-0.00	0.720+/-0.00	0.260+/-0.00	0.460	0.230+/-0.00	0.297+/-0.00
MLPClassifier	324.704+/-0.00	0.748+/-0.00	0.250+/-0.00	0.498	0.170+/-0.00	0.473+/-0.00
GaussianNB	0.029+/-0.00	0.543+/-0.00	0.228+/-0.00	0.315	0.173+/-0.00	0.337+/-0.00

Figure 22- General Models Performance with Over Sampling after improvement

```
Best Threshold=0.436825, F-Score=0.265
```



```
Best Threshold=0.564321, F-Score=0.253
```

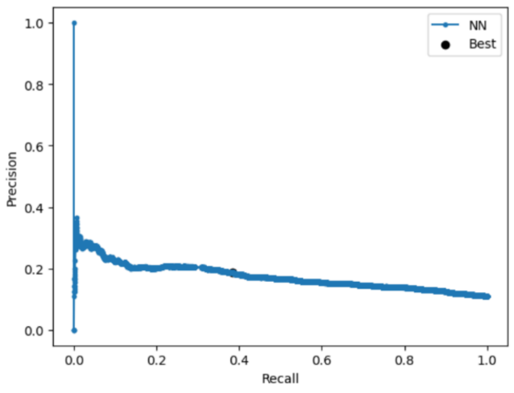
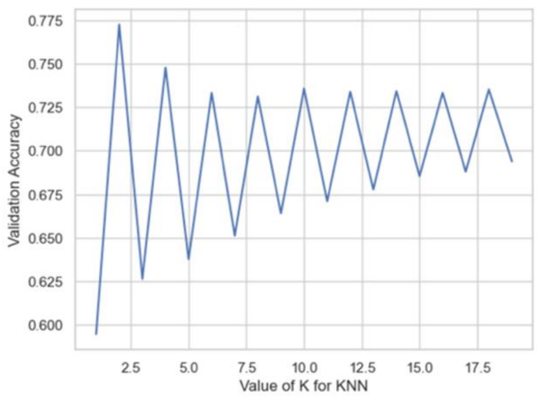


Figure23- Over sampled Threshold

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	0.132+/-0.00	0.592+/-0.00	0.233+/-0.00	0.359	0.205+/-0.00	0.272+/-0.00
KNeighborsClassifier	0.049+/-0.00	0.729+/-0.00	0.207+/-0.00	0.522	0.139+/-0.00	0.406+/-0.00
DecisionTreeClassifier	0.106+/-0.00	0.999+/-0.00	0.213+/-0.00	0.786	0.137+/-0.00	0.483+/-0.00
MLPClassifier	24.124+/-0.00	0.687+/-0.00	0.273+/-0.00	0.414	0.197+/-0.00	0.442+/-0.00
GaussianNB	0.003+/-0.00	0.448+/-0.00	0.191+/-0.00	0.258	0.202+/-0.00	0.180+/-0.00

Figure 24- General Models Performance with Under Sampling



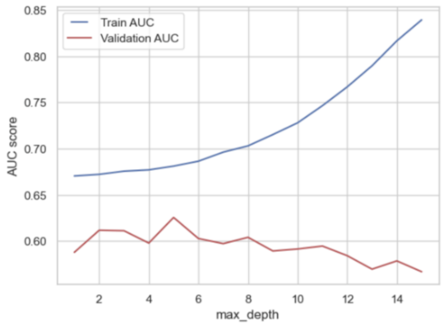
```
The best value for n_neighbors is 2 with accuracy 0.77

Best Parameters: {'algorithm': 'auto', 'n_neighbors': 2,

'p': 2, 'weights': 'distance'}

F1 Score to the Train dataset: 0.9992960500586625
F1 Score to the Validation dataset: 0.20868306801736614
kaggle result:0.2049 version 38
```

Figure 25- KNN Under Sampling best parameters

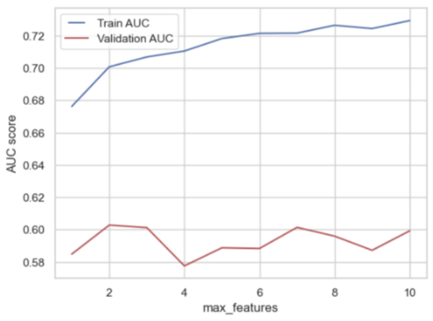
```
Best Parameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'sag'}

F1 Score to the Train dataset: 0.5985680190930788
F1 Score to the Validation dataset: 0.22495768579781505
kaggle result: 0.2439 version 39
```

Figure 26- Logistic Regression Under Sampling best parameters

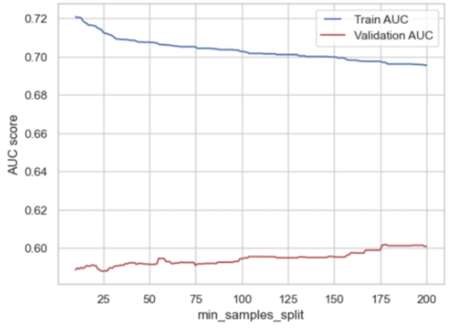
```
The best train value is  15
The best validation value is  5
```



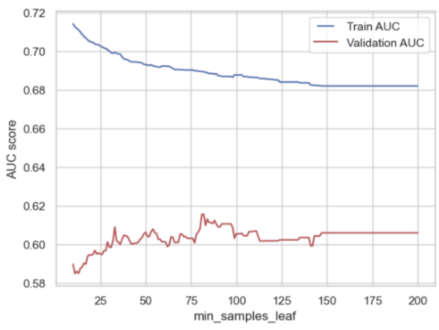
```
The best train value is  10
The best validation value is  2
```



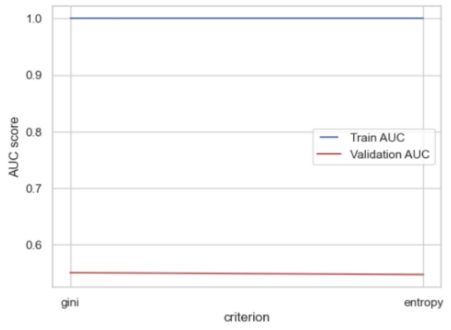
```
The best train value is  10
The best validation value is  178
```



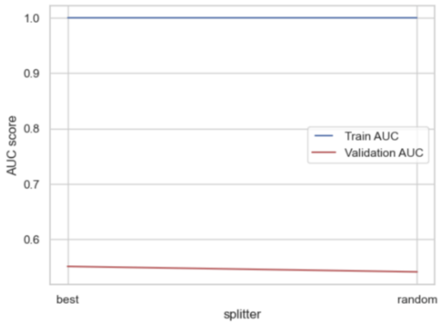
```
The best train value is  10
The best validation value is  82
```



```
The best train value is  gini
The best validation value is  gini
```



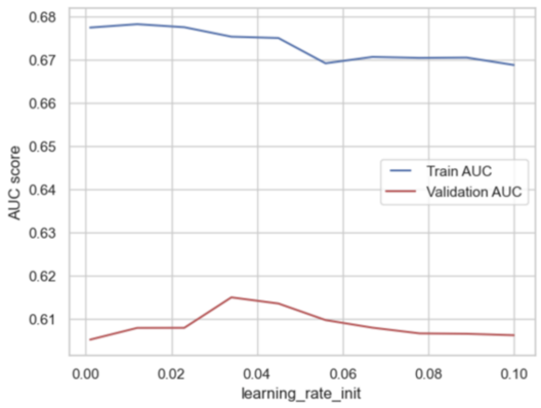
```
The best train value is  best
The best validation value is  best
```



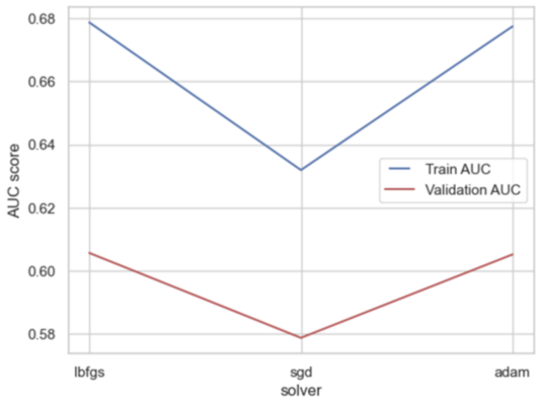
```
F1 Score to the Train dataset: 0.6905103969754253
F1 Score to the Validation dataset: 0.27448609431680776
kaggle result: 0.3016 version 70
```

Figure 27- Decision Tree Under Sampling best parameters

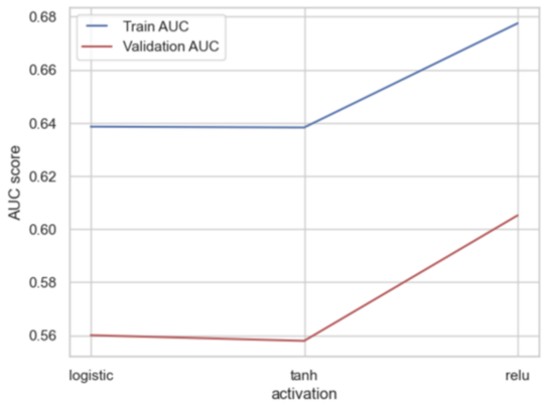
```
The best train value is  0.012
The best validation value is  0.034
```



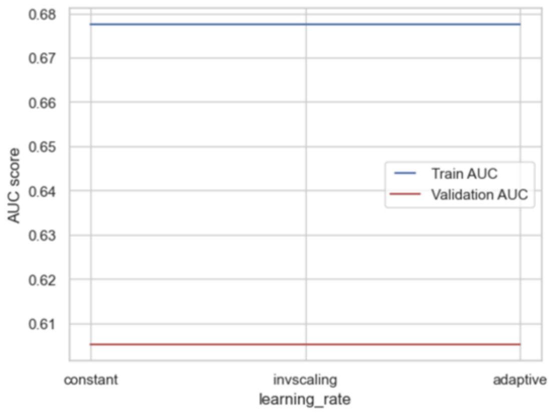
```
The best train value is  lbfgs
The best validation value is  lbfgs
```



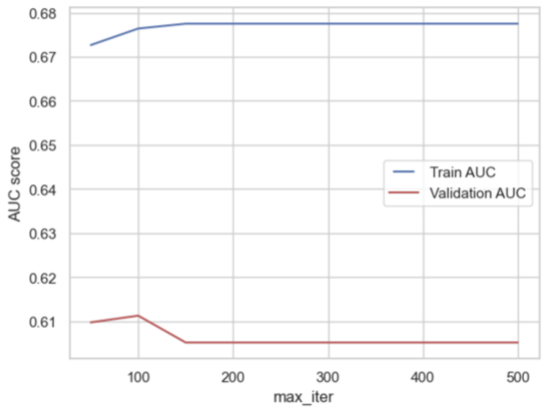
```
The best train value is  relu
The best validation value is  relu
```



```
The best train value is  constant
The best validation value is  constant
```



```
The best train value is  150
The best validation value is  100
```



```
Best Parameters: {'activation': 'relu', 'hidden_layer_sizes': 8,
'constant', 'learning_rate_init': 0.034, 'solver': 'lbfgs'}

F1 Score to the Train dataset: 0.6886799437328792
F1 Score to the Validation dataset: 0.2650432327510146
kaggle result:0.2844 version71
```

Figure 28- Neural Networks Under Sampling best parameters

```
Best Parameters: {'var_smoothing': 1e-09}

F1 Score to the Train dataset: 0.4579845289816679
F1 Score to the Validation dataset: 0.1946264744429882
kaggle result:0.2294 version42
```

Figure 29- Naïve Bayes Under Sampling best parameters

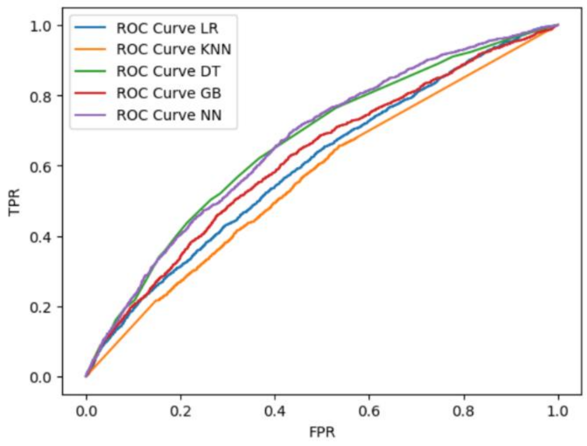
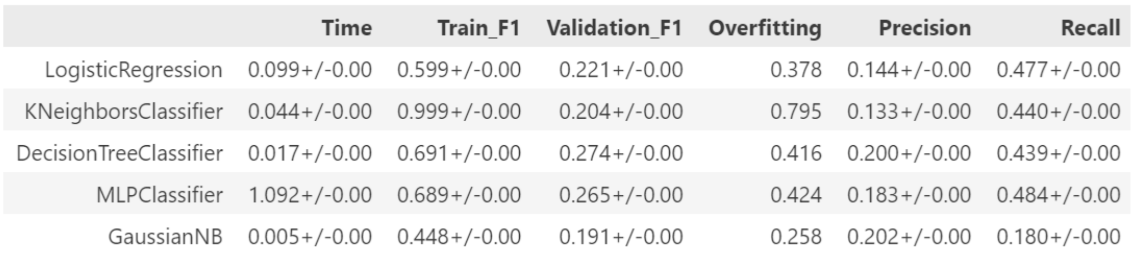
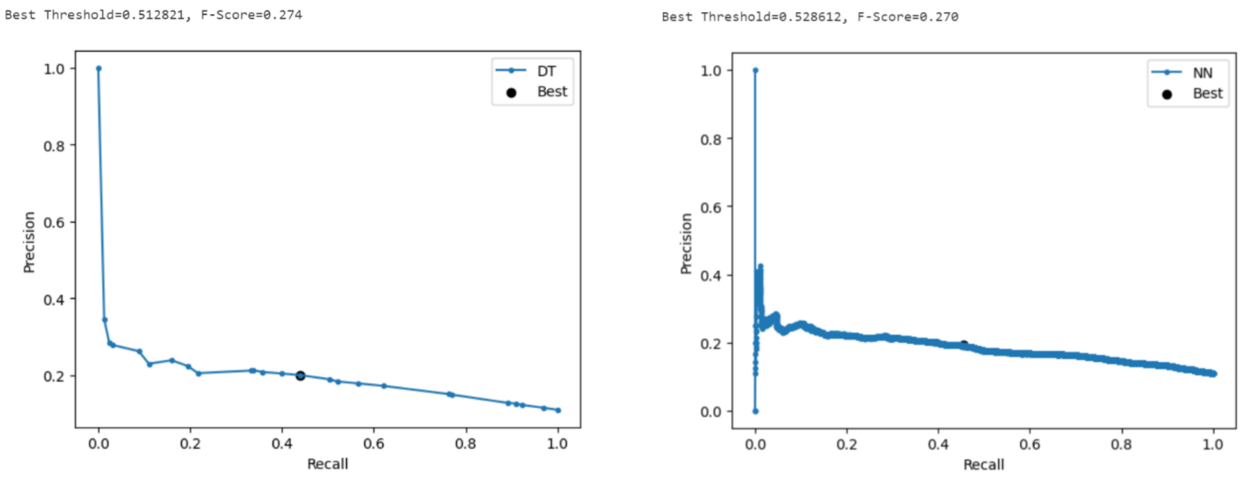


Figure 30- ROC curve for under sampling



	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	0.099+/-0.00	0.599+/-0.00	0.221+/-0.00	0.378	0.144+/-0.00	0.477+/-0.00
KNeighborsClassifier	0.044+/-0.00	0.999+/-0.00	0.204+/-0.00	0.795	0.133+/-0.00	0.440+/-0.00
DecisionTreeClassifier	0.017+/-0.00	0.691+/-0.00	0.274+/-0.00	0.416	0.200+/-0.00	0.439+/-0.00
MLPClassifier	1.092+/-0.00	0.689+/-0.00	0.265+/-0.00	0.424	0.183+/-0.00	0.484+/-0.00
GaussianNB	0.005+/-0.00	0.448+/-0.00	0.191+/-0.00	0.258	0.202+/-0.00	0.180+/-0.00

Figure 31- General Models Performance with Under Sampling after improvement

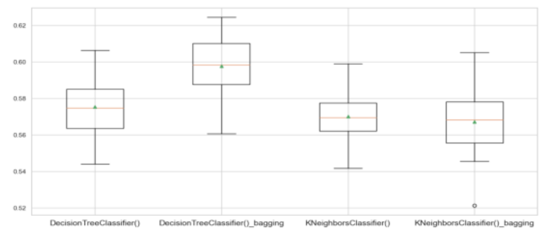


```
Best Threshold=0.512821, F-Score=0.274
```

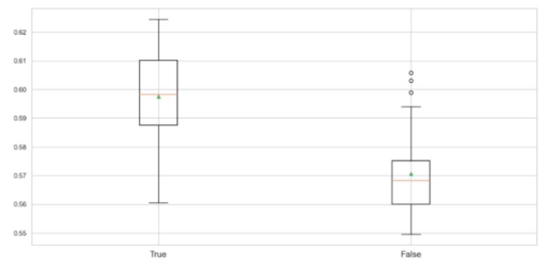
```
Best Threshold=0.528612, F-Score=0.270
```

Figure 32- Under sample Threshold DT, NN

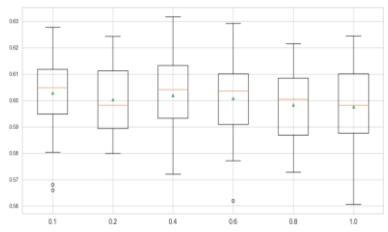
```
>DecisionTreeClassifier() 0.575 (0.015)
>DecisionTreeClassifier()_bagging 0.598 (0.017)
>KNeighborsClassifier() 0.570 (0.012)
>KNeighborsClassifier()_bagging 0.567 (0.018)
```



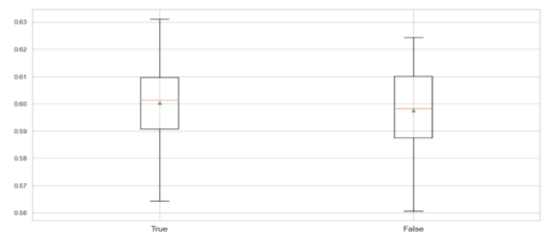
```
>True 0.598 (0.017)
>False 0.571 (0.015)
```



```
>0.1 0.603 (0.015)
>0.2 0.600 (0.013)
>0.4 0.602 (0.015)
>0.6 0.601 (0.015)
>0.8 0.598 (0.013)
>1.0 0.598 (0.017)
```



```
>True 0.600 (0.014)
>False 0.598 (0.017)
```



```
>2 0.456 (0.018)
>5 0.608 (0.015)
>10 0.598 (0.017)
>20 0.623 (0.014)
>30 0.631 (0.015)
>50 0.641 (0.013)
>100 0.647 (0.012)
```

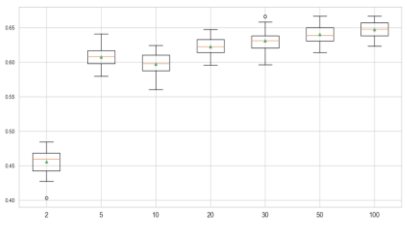
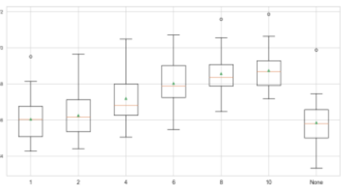
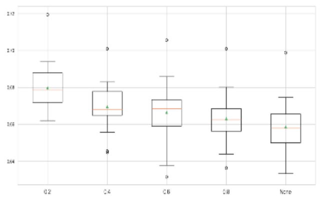


Figure 33- Bagging hyperparameters, classifier, max_samples, n_estimators, bootstrap and bootstrap_feature

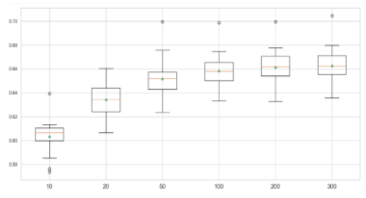
```
>1 0.660 (0.012)
>2 0.663 (0.012)
>4 0.672 (0.012)
>6 0.680 (0.012)
>8 0.686 (0.011)
>10 0.687 (0.010)
>None 0.659 (0.013)
```



```
>0.2 0.680 (0.012)
>0.4 0.670 (0.011)
>0.6 0.667 (0.014)
>0.8 0.663 (0.012)
>None 0.659 (0.013)
```



```
>10 0.604 (0.012)
>20 0.634 (0.013)
>50 0.652 (0.015)
>100 0.659 (0.013)
>200 0.662 (0.013)
>300 0.663 (0.013)
```



```
>True 0.659 (0.013)
>False 0.645 (0.013)
```

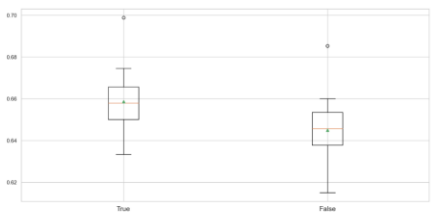
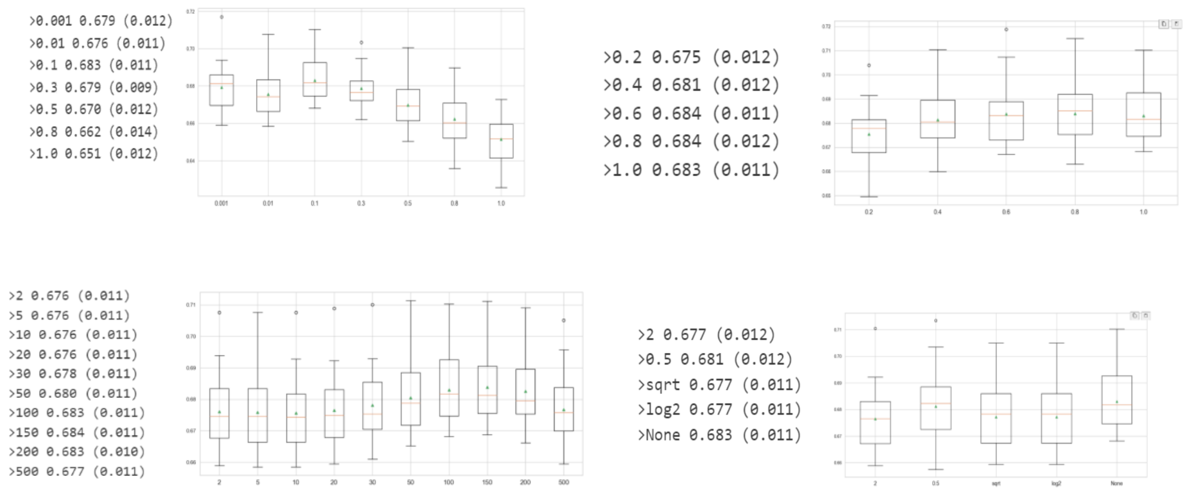


Figure 34- Random Forest, max_depth, n_estimators, max_samples, bootstrap



```
>0.001 0.679 (0.012)
>0.01 0.676 (0.011)
>0.1 0.683 (0.011)
>0.3 0.679 (0.009)
>0.5 0.670 (0.012)
>0.8 0.662 (0.014)
>1.0 0.651 (0.012)
```

```
>0.2 0.675 (0.012)
>0.4 0.681 (0.012)
>0.6 0.684 (0.011)
>0.8 0.684 (0.012)
>1.0 0.683 (0.011)
```

```
>2 0.676 (0.011)
>5 0.676 (0.011)
>10 0.676 (0.011)
>20 0.676 (0.011)
>30 0.678 (0.011)
>50 0.680 (0.011)
>100 0.683 (0.011)
>150 0.684 (0.011)
>200 0.683 (0.010)
>500 0.677 (0.011)
```

```
>2 0.677 (0.012)
>0.5 0.681 (0.012)
>sqrt 0.677 (0.011)
>log2 0.677 (0.011)
>None 0.683 (0.011)
```

Figure 35- Gradient Boosting, learning_rate, n_estimators, subsample, max_features

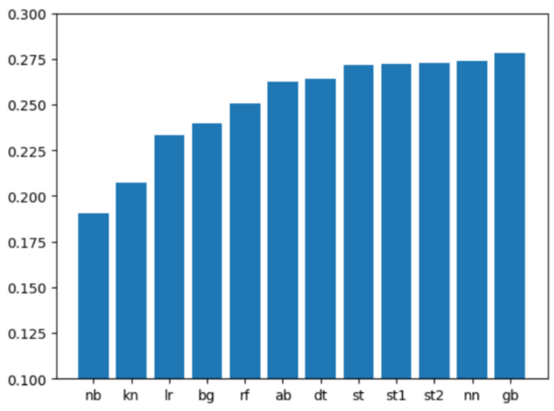
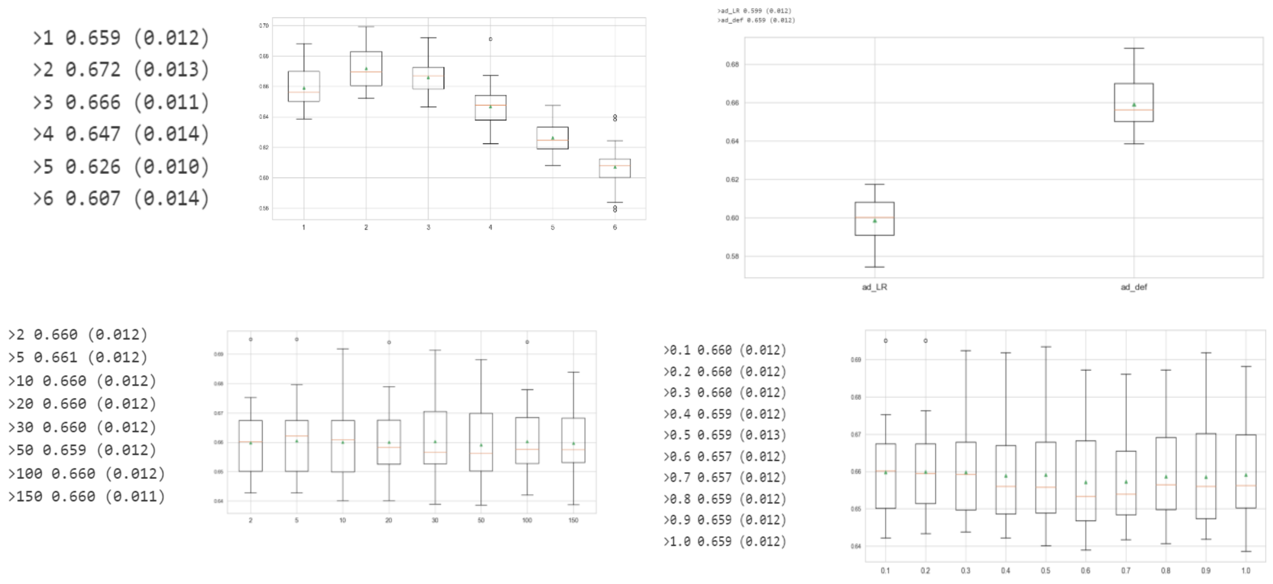


Figure 36- Stacking



```
>1 0.659 (0.012)
>2 0.672 (0.013)
>3 0.666 (0.011)
>4 0.647 (0.014)
>5 0.626 (0.010)
>6 0.607 (0.014)
```

```
>ad_LR 0.599 (0.012)
>ad_def 0.659 (0.012)
```

```
>2 0.660 (0.012)
>5 0.661 (0.012)
>10 0.660 (0.012)
>20 0.660 (0.012)
>30 0.660 (0.012)
>50 0.659 (0.012)
>100 0.660 (0.012)
>150 0.660 (0.011)
```

```
>0.1 0.660 (0.012)
>0.2 0.660 (0.012)
>0.3 0.660 (0.012)
>0.4 0.659 (0.012)
>0.5 0.659 (0.013)
>0.6 0.657 (0.012)
>0.7 0.657 (0.012)
>0.8 0.659 (0.012)
>0.9 0.659 (0.012)
>1.0 0.659 (0.012)
```

Figure 37- AdaBoost

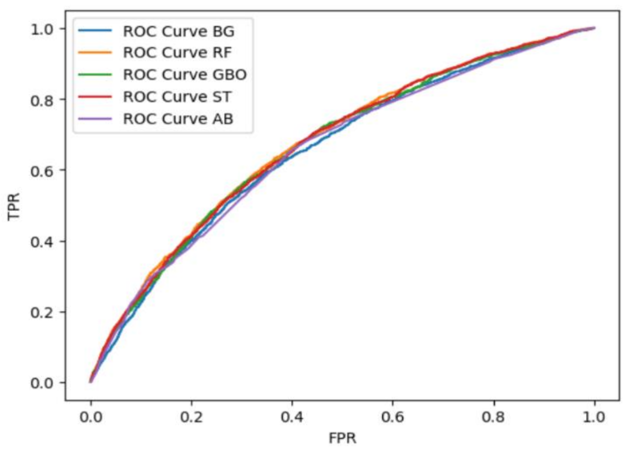
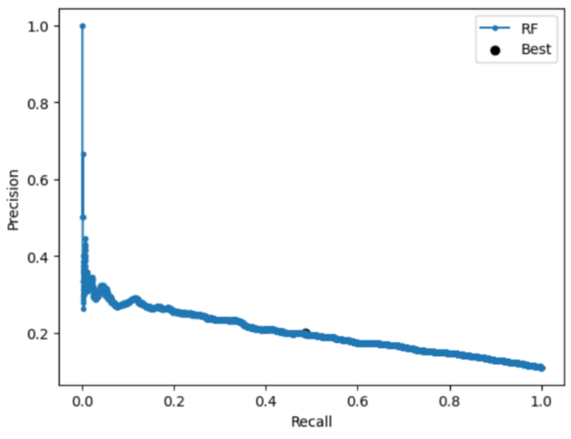
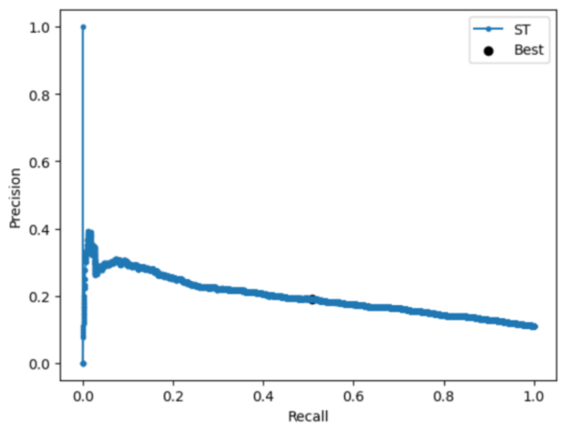


Figure 38- ROC curve ensembles

```
Best Threshold=0.487186, F-Score=0.281
```



```
Best Threshold=0.482596, F-Score=0.278
```



```
Best Threshold=0.496554, F-Score=0.280
```

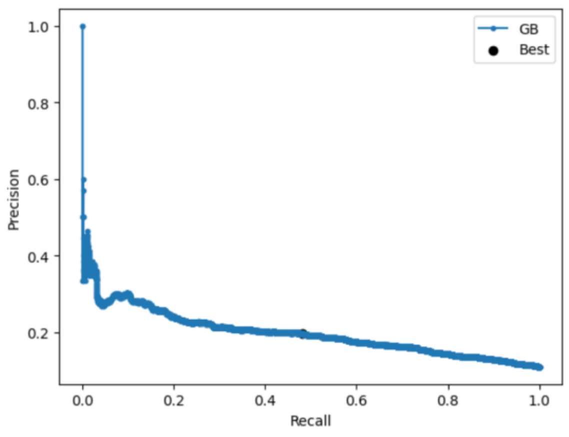


Figure 39- Under sample Threshold

```python
y_train_multiclass = y_train_multiclass.reset_index(drop=True)
y_val_multiclass = y_val_multiclass.reset_index(drop=True)

y_train_multiclass.replace('No', 0, inplace=True)
y_train_multiclass.replace('>30 days', 1, inplace=True)
y_train_multiclass.replace('<30 days', 2, inplace=True)

y_val_multiclass.replace('No', 0, inplace=True)
y_val_multiclass.replace('>30 days', 1, inplace=True)
y_val_multiclass.replace('<30 days', 2, inplace=True)
```

```python
encoder2 = ce.TargetEncoder(cols=non_metric_features)

df_selection_multi = encoder2.fit_transform(df_selection, y_train_multiclass)
df_selection_val_multi = encoder2.transform(df_selection_val)
```

Figure 40- Multiclass Encoding

```
Discard race from the model.

Discard admission_type from the model.

Discard admission_source from the model.

Discard discharge_category from the model.

Discard Exams_performed_results from the model.

Discard new_medications from the model.
```

	Feature	Mutual_Information
0	admission_source	0.011255
1	admission_type	0.008939
2	Exams_performed_results	0.006498
3	new_medications	0.006304
4	discharge_category	0.004317
5	race	0.004102

Figure 41- Feature Selection using Chi-Squared and Mutual Information

	Feature	RFE Logistic	RFE DT	Lasso	Ridge	Decision Tree	ANOVA	Decision
0	gender	Discard	Discard	Keep	Discard	Discard	Keep	Discard
1	age	Keep	Discard	Keep	Discard	Keep	Keep	Keep
2	average_pulse_bpm	Discard	Keep	Keep	Discard	Keep	Discard	Keep
3	number_diagnoses	Keep	Discard	Keep	Discard	Discard	Keep	Keep
4	change_in_meds_during_hospitalization	Discard	Discard	Keep	Discard	Discard	Keep	Discard
5	prescribed_diabetes_meds	Discard	Discard	Keep	Discard	Discard	Keep	Discard
6	total_procedures	Discard	Keep	Keep	Discard	Keep	Keep	Keep
7	Insurance_or_not	Discard	Discard	Keep	Discard	Discard	Discard	Discard
8	total_points	Discard	Keep	Keep	Discard	Keep	Keep	Keep
9	total_visits	Keep	Discard	Keep	Keep	Discard	Keep	Keep
10	medications_per_day	Keep	Keep	Keep	Discard	Keep	Keep	Keep
11	Number_encounter	Keep	Keep	Keep	Keep	Keep	Keep	Keep

Figure 42- Multiclass Feature Selection

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	1.465+/-0.00	0.494+/-0.00	0.507+/-0.00	-0.013	0.556+/-0.00	0.579+/-0.00
KNeighborsClassifier	0.261+/-0.00	0.805+/-0.00	0.440+/-0.00	0.364	0.465+/-0.00	0.423+/-0.00
DecisionTreeClassifier	2.224+/-0.00	1.000+/-0.00	0.477+/-0.00	0.523	0.471+/-0.00	0.486+/-0.00
MLPClassifier	518.522+/-0.00	0.579+/-0.00	0.537+/-0.00	0.042	0.545+/-0.00	0.575+/-0.00
GaussianNB	0.036+/-0.00	0.411+/-0.00	0.485+/-0.00	-0.074	0.546+/-0.00	0.561+/-0.00

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	2.254+/-0.00	0.496+/-0.00	0.505+/-0.00	-0.009	0.556+/-0.00	0.580+/-0.00
KNeighborsClassifier	0.273+/-0.00	1.000+/-0.00	0.463+/-0.00	0.537	0.457+/-0.00	0.473+/-0.00
DecisionTreeClassifier	0.118+/-0.00	0.435+/-0.00	0.422+/-0.00	0.013	0.346+/-0.00	0.544+/-0.00
MLPClassifier	13.217+/-0.00	0.443+/-0.00	0.443+/-0.00	0.001	0.475+/-0.00	0.538+/-0.00
GaussianNB	0.028+/-0.00	0.411+/-0.00	0.485+/-0.00	-0.074	0.546+/-0.00	0.561+/-0.00

Figure 43- Multiclass Over sample Models, without and with parameters

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	0.384+/-0.00	0.481+/-0.00	0.508+/-0.00	-0.027	0.550+/-0.00	0.572+/-0.00
KNeighborsClassifier	0.047+/-0.00	0.586+/-0.00	0.458+/-0.00	0.128	0.463+/-0.00	0.457+/-0.00
DecisionTreeClassifier	0.271+/-0.00	1.000+/-0.00	0.446+/-0.00	0.554	0.480+/-0.00	0.426+/-0.00
MLPClassifier	24.513+/-0.00	0.502+/-0.00	0.522+/-0.00	-0.021	0.553+/-0.00	0.561+/-0.00
GaussianNB	0.005+/-0.00	0.405+/-0.00	0.476+/-0.00	-0.071	0.532+/-0.00	0.560+/-0.00

	Time	Train_F1	Validation_F1	Overfitting	Precision	Recall
LogisticRegression	0.384+/-0.00	0.481+/-0.00	0.508+/-0.00	-0.027	0.550+/-0.00	0.572+/-0.00
KNeighborsClassifier	0.047+/-0.00	0.586+/-0.00	0.458+/-0.00	0.128	0.463+/-0.00	0.457+/-0.00
DecisionTreeClassifier	0.271+/-0.00	1.000+/-0.00	0.446+/-0.00	0.554	0.480+/-0.00	0.426+/-0.00
MLPClassifier	24.513+/-0.00	0.502+/-0.00	0.522+/-0.00	-0.021	0.553+/-0.00	0.561+/-0.00
GaussianNB	0.005+/-0.00	0.405+/-0.00	0.476+/-0.00	-0.071	0.532+/-0.00	0.560+/-0.00

Figure 44- Multiclass Under sample Models, without and with parameters

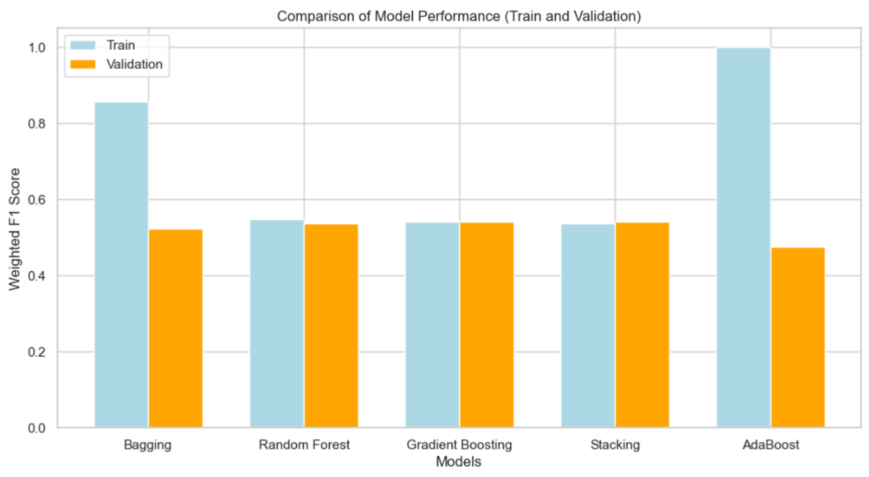
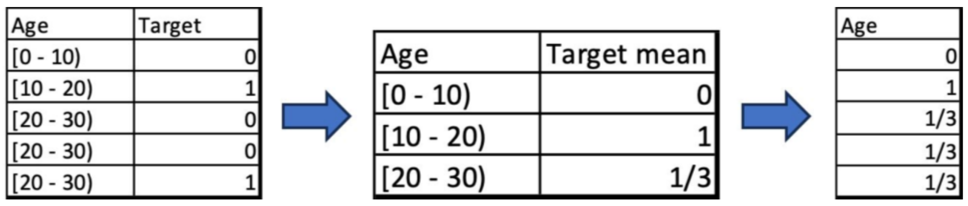


Figure 45- Comparison of ensembled models

CREATIVITY AND OTHER SELF-STUDIES

1. Target Encoding

Most models do not work well with categorical variables so these must be changed into numerical. Target Encoder (calculates the average target values for a specific category and then uses it as a category) is an approach that provides a solution to this issue and avoids facing the Dimensionality's curse (meaning that are too many categories making it harder to establish patterns).



Age	Target
[0 - 10)	0
[10 - 20)	1
[20 - 30)	0
[20 - 30)	0
[20 - 30)	1

Age	Target mean
[0 - 10)	0
[10 - 20)	1
[20 - 30)	1/3

Age
0
1
1/3
1/3
1/3

2. Ridge Regression

Ridge Regression was implemented for Feature Selection, to complement the methods learned in class. This model helps analyse the multicollinearity in multiple regression data, aiming to minimise the sum of squared residuals, the errors between the actual and the predicted values. This tends to be reduced by introducing a penalty term to the regression, lambda, tending to bring the coefficients closer to zero but never gets to zero, as it prevents overfitting and reducing the complexity of the model. In Ridge Regression the features eliminated are the ones where the optimal slope of the model gets closer to zero. In this project, the Ridge Regression was used with the penalty being L2 that penalise the weights of the features coefficients, and with a C=1 that is the inverse of the regularisation strength (alpha) so the higher it is the stronger the regularisation.

3. ANOVA

The ANOVA method was also used for feature selection as it is an Analysis of Variance, by checking the means of groups that are significantly different from each other. It is used to see the impact of the independent variables on the dependent one. With this, can be decided if the null hypothesis (all the sample means are the same) is going to be accepted or not, this is, reject the null hypothesis so the alternative hypothesis is used (at least one of the sample means used are different from the other ones being measured). So, the higher the F-statistic is the less similar the sample means are so the null hypothesis can be rejected. ANOVA calculates the sum of the squares to check the dispersion of the data points and uses the F-statistic and if there is no significant difference that all the variances are equal, the results will be closer to 1.

4. SMOTE and Random Under Sampler

As mentioned along the report the datasets being used are imbalanced, so it was required to resampling. Over sampling investigates completing the minority class by adding entrances to it, in this case, it was used the SMOTE technique which instead of duplicating instances, it selected some of them which share similar traits and then create a synthetic instance using the distance metrics. On the other hand, under sampling investigates reducing the majority class of the sample by eliminating some of its data entrances. In this project, the random sampling technique was the selected, as it randomly selects transactions and can be repeated until reaching the desired outcome.

5. ROC curve and threshold

The ROC curve (Receiver Operating Characteristic) and the Threshold concepts complement each other on the grounds that the first is the trade-off between the true positives with the false negatives, while the second identifies the point of intersection between these two parameters, balancing both specify and sensitivity. Analysing both together provides a better sense of decision regarding the model performance. The model with the best results not only has a superior threshold but is also positioned above all others in the ROC curve, beating them with a higher percentage of true and false positives. This means a better prediction as well as more correct results or targets.