



ISLAMIC UNIVERSITY OF TECHNOLOGY

COURSE NO. : EEE 4518

COURSE NAME : Electrical and Electronic Workshop

PROJECT TITLE : Prediction of Mortality among ICU
admitted patients using Machine Learning Algorithms

TEAM NAME : Team Quantaphy

TEAM MEMBERS : 1. Mizanur Rahman - 190021109
2. Redwan-Ul-Bari - 190021119
3. Md. Faiyaz Abrar Fahim - 190021137

TABLE OF CONTENTS:

- 1.** *Objective*
- 2.** *Introduction*
- 3.** *Resource Collection*
- 4.** *Previous Works*
- 5.** *Implementation Algorithm*
- 6.** *Project Details*
 1. Imbalance Ration
 2. NaN Value Handling
 3. KNN Imputer
 4. Correlation Heatmap
 5. Oversampling with BorderlineSMOTE
 6. Principal Component Analysis
 7. Performance Parameters Introduction
 8. Various Models and Evaluation
- 7.** *Discussion*
- 8.** *Future Plans*
- 9.** *Conclusions*
- 10.** *References*

Objective:

The predictors of in-hospital mortality for intensive care units (ICU)-admitted 'Heart Failure' patients remain poorly characterized. The objective of our project is to predict the mortality of ICU admitted Heart failure patients on the basis of different vital counts. We aimed to develop and validate a prediction model using Machine Learning Algorithms. We observe how different training models fare with the provided bio-medical data.

Introduction:

Intensive Care Unit (ICU), is a hospital service specialized for patients who are facing severe health implications and is in need of immediate special care. Patients receive ICU treatment when their health condition is critical and normal hospital care is insufficient for the improvement of the patient's concurrent state. The mortality rate of ICU patients is fairly high specially alarming for aged patients. ICU mortality rate differs greatly depending on the underlying disease and past medical records.

The risk of death due to heart failure in ICU can be predicted by collecting enough data and training machine learning models. Main focus of data collection is the vital counts which are constantly monitored in an ICU. We will be trying different algorithms and try to find which model can get us the most accuracy based on the data that we provide to them.

Situation of a patient in an ICU can deteriorate any time so it is very crucial to take quick decision. Our machine learning model can work as a decision support system. Prediction from ML models coupled with the knowledge and experience of respected doctors' can be used to take precautionary measures.

Resource Collection:

1. Collected from Kaggle

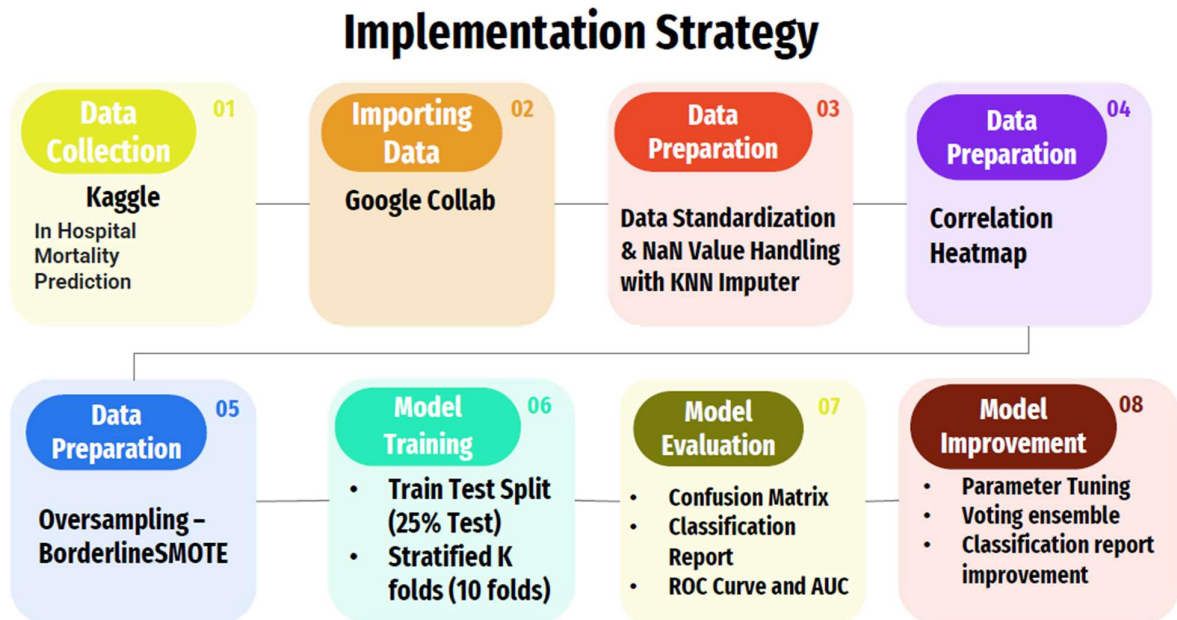
Source: <https://www.kaggle.com/datasets/saurabhshahane/in-hospital-mortality-prediction?resource=download>

Data Set Characteristics	Binary Classification	Data Types	Integer, Float
Number of Instances	1177	Number of Attributes	51
Missing Values	Yes	Associated Task	Classification

Previous Works:

1. Prediction model of in-hospital mortality in intensive care unit patients with heart failure: machine learning-based, retrospective analysis of the MIMIC-III database.^[1]
2. Mortality prediction of patients in intensive care units using machine learning algorithms based on electronic health records.^[2]
3. Early hospital mortality prediction using vital signals.^[3]
4. Machine Learning-based Short-term Mortality Prediction Models for Cancer Patients Using Electronic Health Record Data: A Systematic Review and Critical Appraisal.^[4]
5. Using machine learning methods to predict in-hospital mortality of sepsis patients in the ICU.^[5]
6. Comparing machine learning algorithms for predicting ICU admission and mortality in COVID-19.^[6]
7. Predicting Mortality in Diabetic ICU Patients Using Machine Learning and Severity Indices.^[7]
8. Dynamic and explainable machine learning prediction of mortality in patients in the intensive care unit: a retrospective study of high-frequency data in electronic patient records.^[8]

Implementation Algorithm:



Project Details:

At the very beginning of our project, we imported all the necessary libraries. Then we loaded the dataset and used some simple codes to analyze the dataset information (size, shape, data count, column count, missing value count etc.).

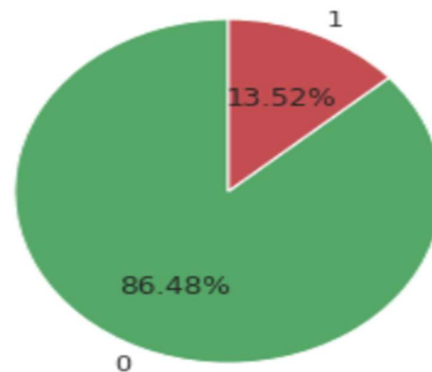
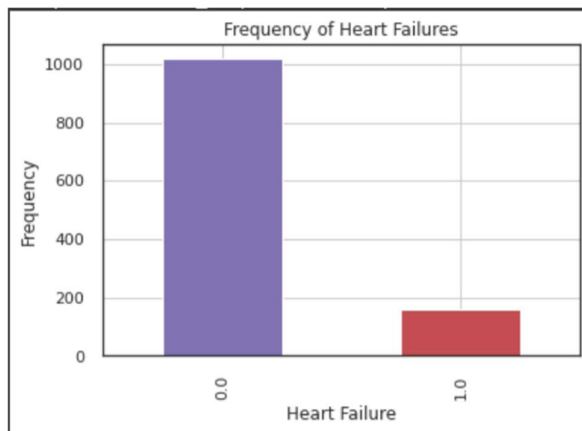
Our dataset contains 1177 rows or instances and 51 columns or features. All the entries were not complete so there existed null values in the dataset. The problem we are trying to solve is a classification problem with the target column 'outcome' which contains the binary information for dead and alive patients. In our 'outcome' column – 0 signifies alive and 1 signifies dead and also there was one entry containing null value so we removed the entire instance.

Imbalance ratio:

It is customary to use a ratio to express the imbalance of classes in a dataset. Our dataset is originally imbalanced and we are unable to obtain a balanced dataset. Here, we can see that the number of positive values is 159, while the number of negative values is 1017.

```
0.0    1017
1.0     159
Name: outcome, dtype: int64
r = 6.39622641509434
```

Unevenly distributed observations throughout the target class, with one class label having a very high number of observations and the other having a very low number, are referred to as imbalanced data. Data collection is generally difficult or expensive, and we frequently gather and use much less data than we would want.



NAN value handling:

Frequently, there are many missing values in the real-world data. The "missing value" portion of your data must be ignored if we want your model to operate objectively and accurately. Dealing with missing values is one of the most prevalent issues in data cleansing or preprocessing.

Data Frame and numpy arrays use the special value NaN, or Not a Number, to indicate when a value is absent from a cell. Columns without observations are considered to have missing data. It is represented by values like 0, NA, and NAN. The outcomes of machine learning models can be skewed or the model's accuracy can be reduced, which is one of the major effects of missing data. Handling missing values is therefore crucial.

Impute missing values with KNN Imputation:

Rows or columns with null values can be removed to manage missing values. Columns can be completely dropped if more than half of their rows are null. You can also remove the rows with one or more columns with null values. However, there are 51 columns and 1177 entities in our data collection. It's a modest data collection, and removing several rows and columns would make it even smaller.

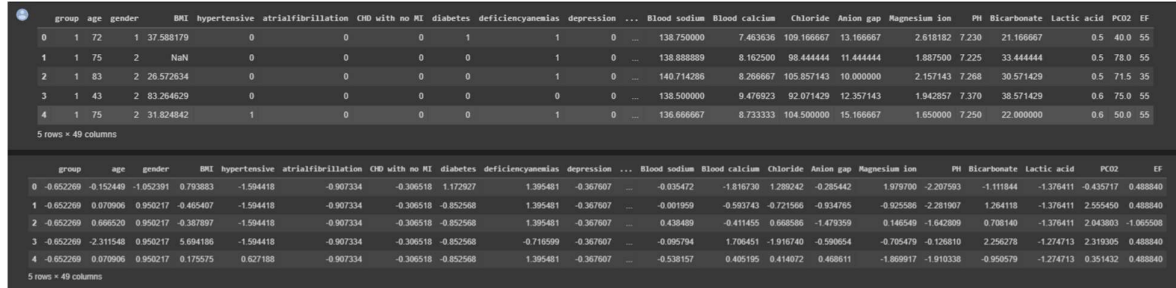
```
[ ] pd.isnull(df).sum()
Group          0
ID              0
outcome        1
age            0
gender         0
BMI           215
hypertensive   0
atrialfibrillation 0
CHD with no MI 0
diabetes       0
deficiencyanemias 0
depression     0
Hyperlipemia   0
Renal failure  0
COPD           0
heart rate     13
Systolic blood pressure 16
Diastolic blood pressure 16
Respiratory rate 13
temperature    19
SP_O2          13
Urine output   36
hematocrit     0
RBC            0
MCH            0
MCHC           0
MCV            0
RDW            0
Leucocyte      0
Platelets      0
Neutrophils    144
Basophils      259
Lymphocyte     145
PT             20
INR            20
NT-proBNP      0
Creatine kinase 165
Creatinine     0
Urea nitrogen  0
glucose        18
Blood potassium 0
Blood sodium   0
Blood calcium  1
Chloride       0
Anion gap      0
Magnesium ion  0
PH             292
Bicarbonate     0
Lactic acid     229
PCO2            294
EF              0
dtype: int64
```

```
Group          0.000000
ID              0.000000
outcome        0.084962
age            0.000000
gender         0.000000
BMI           18.266780
hypertensive   0.000000
atrialfibrillation 0.000000
CHD with no MI 0.000000
diabetes       0.000000
deficiencyanemias 0.000000
depression     0.000000
Hyperlipemia   0.000000
Renal failure  0.000000
COPD           0.000000
heart rate     1.104503
Systolic blood pressure 1.359388
Diastolic blood pressure 1.359388
Respiratory rate 1.104503
temperature    1.614274
SP_O2          1.104503
Urine output   3.058624
hematocrit     0.000000
RBC            0.000000
MCH            0.000000
MCHC           0.000000
MCV            0.000000
RDW            0.000000
Leucocyte      0.000000
Platelets      0.000000
Neutrophils    12.234494
Basophils      22.005098
Lymphocyte     12.319456
PT             1.699235
INR            1.699235
NT-proBNP      0.000000
Creatine kinase 14.018692
Creatinine     0.000000
Urea nitrogen  0.000000
glucose        1.529312
Blood potassium 0.000000
Blood sodium   0.000000
Blood calcium  0.084962
Chloride       0.000000
Anion gap      0.000000
Magnesium ion  0.000000
PH             24.808836
Bicarbonate     0.000000
Lactic acid     19.456245
PCO2            24.978760
EF              0.000000
dtype: float64
```

For this reason, we will not remove our entries rather we will be imputing the entries with a tool called 'KNN Imputation' technique along with data standardization.

Data Standardization: Data standardization transforms data into a format that is recognized and understood by computers. This is significant because it enables data sharing and efficient data utilization among various systems. It would be difficult for various ways to communicate and exchange information without data standards. It is considerably simpler to spot problems and ensure that data is reliable when it is standardized. Making sure decision-makers have access to accurate and trustworthy information is crucial. To make sure that data is accessible and usable, data standardization is essential. Without it, we wouldn't be able to efficiently manage and use data.

NaN Value Handling with KNN Imputer: In our data set we first standardized our data and then replaced our NaN values using the KNN Imputer.



The image displays two screenshots of a Jupyter Notebook, showing data before and after KNN imputation. The top screenshot shows a dataset with 5 rows and 49 columns. The columns include 'group', 'age', 'gender', 'BMI', 'hypertensive', 'atrialfibrillation', 'CHD with no MI', 'diabetes', 'deficiencymenias', 'depression', 'Blood sodium', 'Blood calcium', 'Chloride', 'Anion gap', 'Magnesium ion', 'PH', 'Bicarbonate', 'Lactic acid', 'PCO2', and 'EF'. The bottom screenshot shows the same dataset after KNN imputation, with the same columns and 5 rows. The values in the cells are now numerical, indicating that the NaN values have been replaced.

	group	age	gender	BMI	hypertensive	atrialfibrillation	CHD with no MI	diabetes	deficiencymenias	depression	...	Blood sodium	Blood calcium	Chloride	Anion gap	Magnesium ion	PH	Bicarbonate	Lactic acid	PCO2	EF
0	1	72	1	37.588179	0	0	0	1	1	0	...	138.750000	7.463636	109.166667	13.166667	2.618182	7.230	21.166667	0.5	40.0	55
1	1	75	2	NaN	0	0	0	0	1	0	...	138.888889	8.162500	98.444444	11.444444	1.887500	7.225	33.444444	0.5	78.0	55
2	1	83	2	26.572634	0	0	0	0	1	0	...	140.714286	8.266667	105.857143	10.000000	2.157143	7.268	30.571429	0.5	71.5	35
3	1	43	2	83.264629	0	0	0	0	0	0	...	138.500000	9.476923	92.071429	12.357143	1.942857	7.370	38.571429	0.6	75.0	55
4	1	75	2	31.824842	1	0	0	0	1	0	...	136.666667	8.733333	104.500000	15.166667	1.650000	7.250	22.000000	0.6	50.0	55

5 rows x 49 columns

	group	age	gender	BMI	hypertensive	atrialfibrillation	CHD with no MI	diabetes	deficiencymenias	depression	...	Blood sodium	Blood calcium	Chloride	Anion gap	Magnesium ion	PH	Bicarbonate	Lactic acid	PCO2	EF
0	-0.652269	-0.152449	-1.052391	0.793883	-1.594418	-0.907334	-0.306518	1.172927	1.395481	-0.367607	...	-0.035472	-1.818730	1.289242	-0.285442	1.979700	-2.207593	-1.111844	-1.376411	-0.435717	0.488840
1	-0.652269	0.070906	0.950217	-0.405407	-1.594418	-0.907334	-0.306518	-0.852568	1.395481	-0.367607	...	-0.001959	-0.593743	-0.721566	-0.934705	-0.925586	-2.281907	1.264118	-1.376411	2.555450	0.488840
2	-0.652269	0.666520	0.950217	-0.387897	-1.594418	-0.907334	-0.306518	-0.852568	1.395481	-0.367607	...	-0.438489	-0.411455	0.608586	-1.479359	0.146549	-1.642809	0.708140	-1.376411	2.843803	-1.065508
3	-0.652269	-2.311540	0.950217	5.694186	-1.594418	-0.907334	-0.306518	-0.852568	-0.716599	-0.367607	...	-0.095794	1.706451	-1.916740	-0.590054	-0.705479	-0.126810	2.256278	-1.274713	2.319305	0.488840
4	-0.652269	0.070906	0.950217	0.175575	0.627188	-0.907334	-0.306518	-0.852568	1.395481	-0.367607	...	-0.538157	0.405195	0.414072	0.488611	-1.889917	-1.910338	-0.950579	-1.274713	0.351432	0.488840

5 rows x 49 columns

Correlation Heatmap:

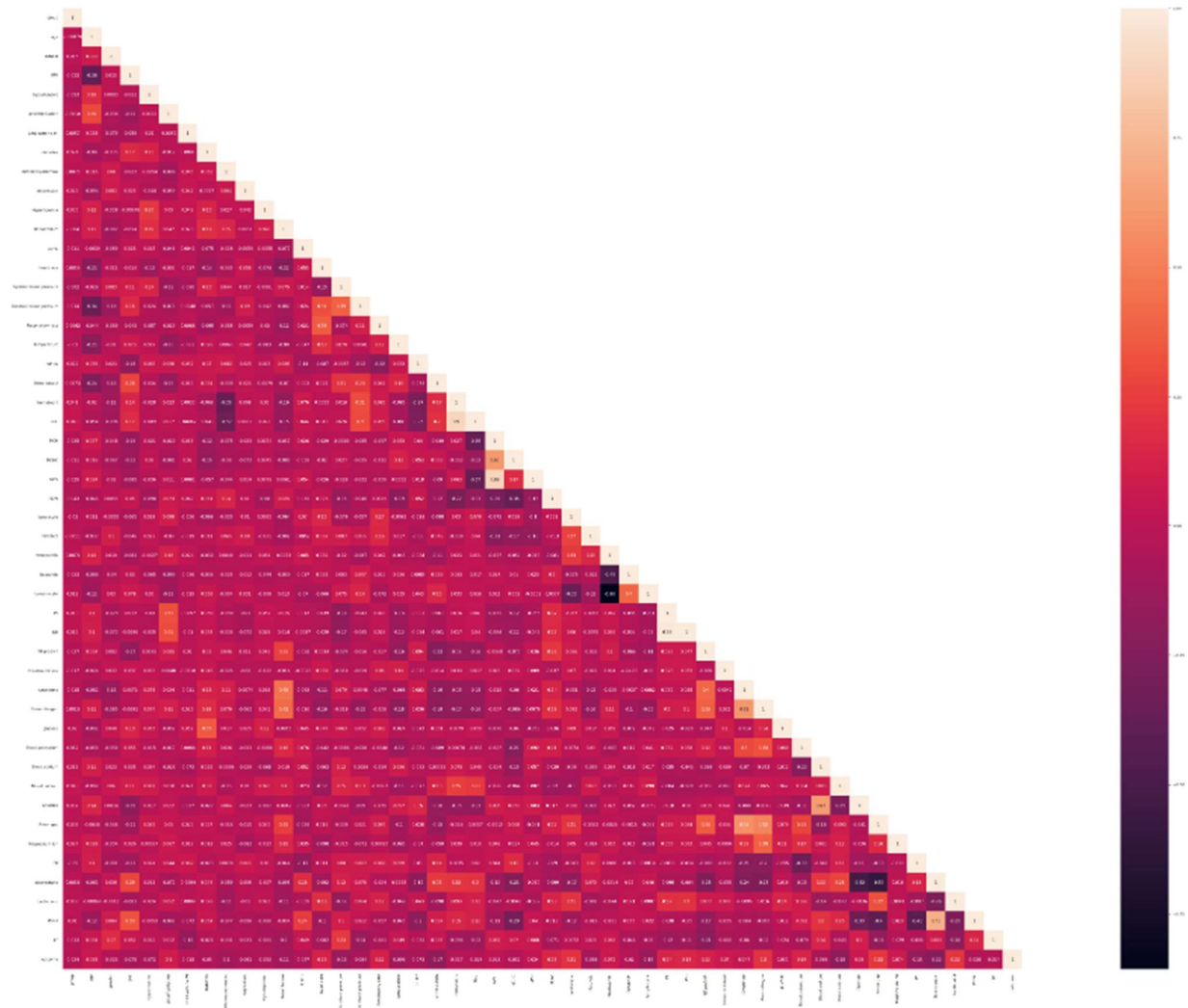
A form of graphic called a correlation heatmap shows the strength of correlations between numerical variables. To determine which variables are related to one another and how strongly this relationship exists, correlation graphs are utilized.

Typically, a correlation plot includes a number of numerical variables, each of which is represented by a column. The relationships between each pair of variables are shown by the rows. Positive values indicate a strong relationship, while negative values indicate a weak relationship. The values in the cells represent the strength of the relationship. We can use correlation heatmaps to identify possible links between variables and to gauge how strong these relationships are. Additionally, outliers can be found and linear and nonlinear correlations can be found using correlation graphs. It is simple to quickly spot relationships between variables because to the cells' color coding. It is a very effective tool when used properly. We can identify highly correlated variables and this will allows us to streamline the feature selection process. The value of the correlation coefficient can take any values from -1 to 1.

A positive correlation between two variables is considered to exist if the value is 1. This implies that while one variable rises, the other variable rises as well.

A negative correlation between the two variables is stated to exist if the value is greater than one. This implies that as one variable rises, the other one falls.

The two variables have no association if the value is 0. This indicates that the variables change in a random way in relation to one another. Black and dark red were used in heatmaps to represent negative or lower value metrics, whereas white and light red were used to represent positive or higher value metrics.



Test and train splitting:

The train-test split is used to gauge how well machine learning algorithms work in applications that rely on predictions.

An area of data is used to fit and train the model and we called it the training dataset. The models will observe and absorb this data.

Then a sample set is used to evaluate the training model and we called it the testing dataset. An accurate assessment of a final model fit is provided by the testing dataset, which is a subset of the training dataset.

In our project the training size was 75% and the testing size was 25%. We use the parameter 'stratify = y' to ensure that the same portion of classes was retained in the train and test dataset as was in the main dataset.

Oversampling using BordelineSMOTE:

To increase the amount of data points in the minority class, we duplicate a random sample of points from the minority class. By lowering the performance requirements for anti-aliasing filters, oversampling can assist prevent aliasing and phase distortion while also increasing resolution and signal-to-noise ratio. If a signal is sampled at N times the Nyquist rate, it is said to be oversampled by a factor of N.

Synthetic Minority Oversampling Technique (SMOTE): We used the imbalanced learn library to use SMOTE a very popular oversampling technique. We used the 'BorderlineSMOTE' a variant of SMOTE where the problem with minority class appearing in the majority class is solved by making the minority class values into noise for cases where the nearest neighbors of the minority class are all majority class.

In our data set we have 763 numbers of 0 values and 119 numbers of 1 values but after oversampling the number of 1 values increases and balances the datasets.

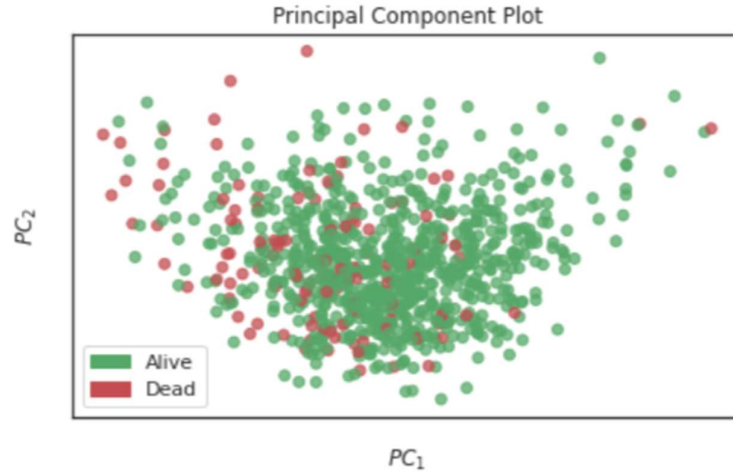
```
Original dataset shape Counter({0.0: 763, 1.0: 119})  
Resampled dataset shape Counter({0.0: 763, 1.0: 763})
```

Principal Component Analysis (PCA):

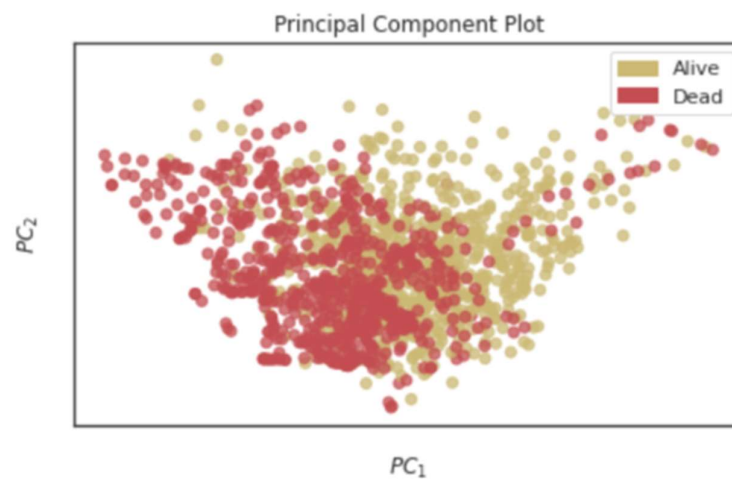
By condensing a huge collection of variables into a smaller one that still contains the majority of the data in the larger set, PCA is a dimensionality-reduction technique that is frequently used to reduce the dimensionality of large data sets. While minimizing information loss, it simultaneously improves interpretability. It makes data easier to plot in 2D and 3D and aids in identifying the dataset's most important properties.

Performance can be enhanced by PCA at a very modest cost to model correctness. The capacity to extract independent, uncorrelated features of the data and the decrease of noise in the data are other advantages of PCA.

PCA plot with split data:



PCA plot with oversampled split data:



Now our data is ready to train and test various classification models. To evaluate the models, we used the following performance parameters:

1. Accuracy of The Model:

For each of the models we calculated their accuracy in terms of the testing data. It is the most basic evaluation of a model.

2. Confusion Matrix:

An in-depth evaluation of a classifier's performance is conducted using a confusion matrix. It helps illustrate the results of a classification task by presenting a table arrangement of the various outcomes of the prediction and findings. It creates a table with all of a classifier's predicted and actual values.

True Positive: The percentage of occasions that our actual positive values match the anticipated positive. You correctly predicted a positive value, which is what it is.

False Positive: The number of times our model incorrectly predicts negative values as positives (or vice versa). In spite of what you had projected, the value is positive.

True Negative: The number of times our actual negative values are equal to predicted negative values. You predicted a negative value, and it is actually negative.

False Negative: The number of times our model wrongly predicts negative values as positives. You predicted a negative value, and it is actually positive.

3. Classification Report:

The accuracy of predictions made by a classification algorithm is evaluated using a classification report. Which predictions came true and which ones didn't. The terms used here includes – precision, recall, scope support etc.

Precision: One measure of a machine learning model's effectiveness is the accuracy of a successful prediction that the model makes. Precision is calculated by dividing the total number of positive predictions by the proportion of genuine positives.

Precision is defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall: The recall is determined as the proportion of Positive samples that were correctly identified as Positive to all Positive samples. The recall gauges how well the model can identify positive samples. The more positive samples that are identified, the larger the recall.

Mathematically, recall is defined

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 – Score: An alternative machine learning evaluation statistic called F1 score evaluates a model's predictive ability by focusing on its performance inside each class rather than its overall performance as is done by accuracy. The F1 score combines two metrics that are in conflict: a model's precision and recall scores, which has led to its extensive adoption in recent research.

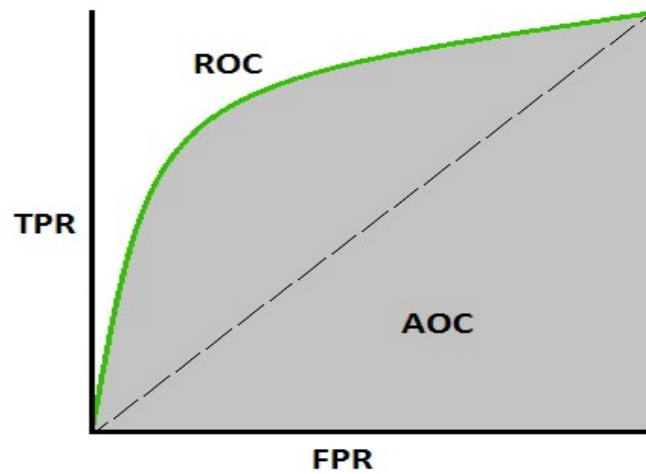
$$F1 \text{ Score} = 2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

4. AUC – ROC Curve:

AUC stands for the level or measurement of separability, and ROC is a probability curve. It reveals how well the model can differentiate across classes. The model is more accurate at classifying 0 classes as 0, and classifying 1 classes as 1 when it has a higher AUC. By analogy, the model is more effective at differentiating between patients with the condition and those who do not have it the higher the AUC. The model performs better at differentiating between the positive and negative classes the higher the AUC. AUC = 1 indicates that the classifier can accurately distinguish between all Positive and Negative class points. However, if the AUC was 0, the classifier would be predicting all negatives as positives, and all positives as negatives.

When $0.5 < \text{AUC} < 1$, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect

more numbers of True positives and True negatives than False negatives and False positives.



Stratified K-Folds:

In our model evaluation we have taken two different approaches. The first approach is to use oversampling technique on ‘Test Train Split’ and the second approach is to use ‘Stratified K-Folds’ on our original data. Using both these we have trained and tested our following models.

Training and Testing Models:

Our problem is a classification problem and we used classifier models for our project.

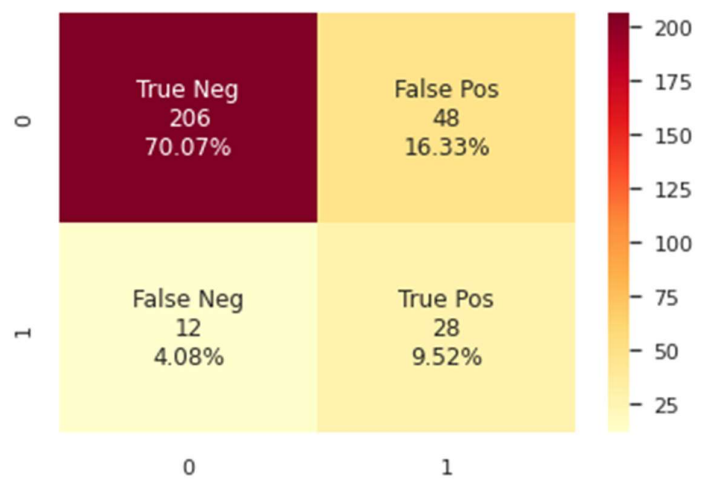
1. Logistic Regression:

When a dependent variable is dichotomous, the proper regression analysis to use is logistic regression (binary). Predictive analysis is what logistic regression is. To describe data and explain the relationship between one dependent binary variable and one or more independent nominal, ordinal, interval, or ratio-level variables, we employ logistic regression.

Accuracy of The Model:

Accuracy of Training Data: 83.22411533420708
Accuracy of Test Data: 79.59183673469387

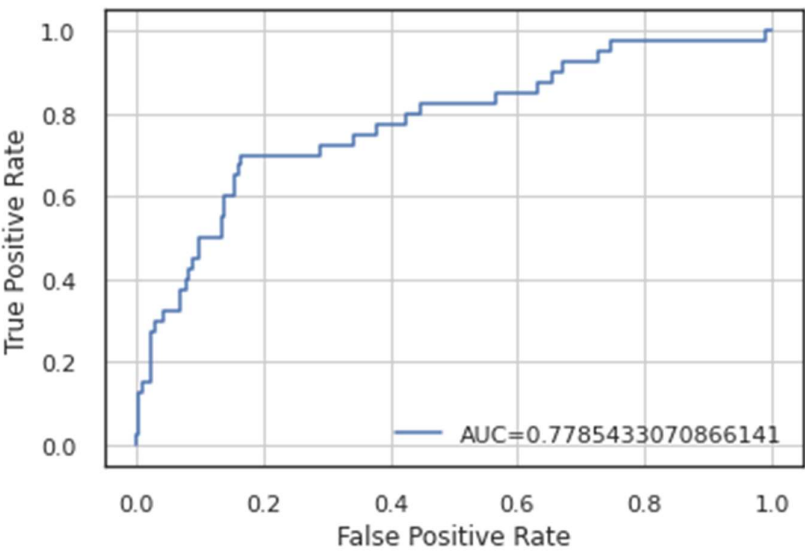
Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0.0	0.94	0.81	0.87	254
1.0	0.37	0.70	0.48	40
accuracy			0.80	294
macro avg	0.66	0.76	0.68	294
weighted avg	0.87	0.80	0.82	294

AUC – ROC Curve:

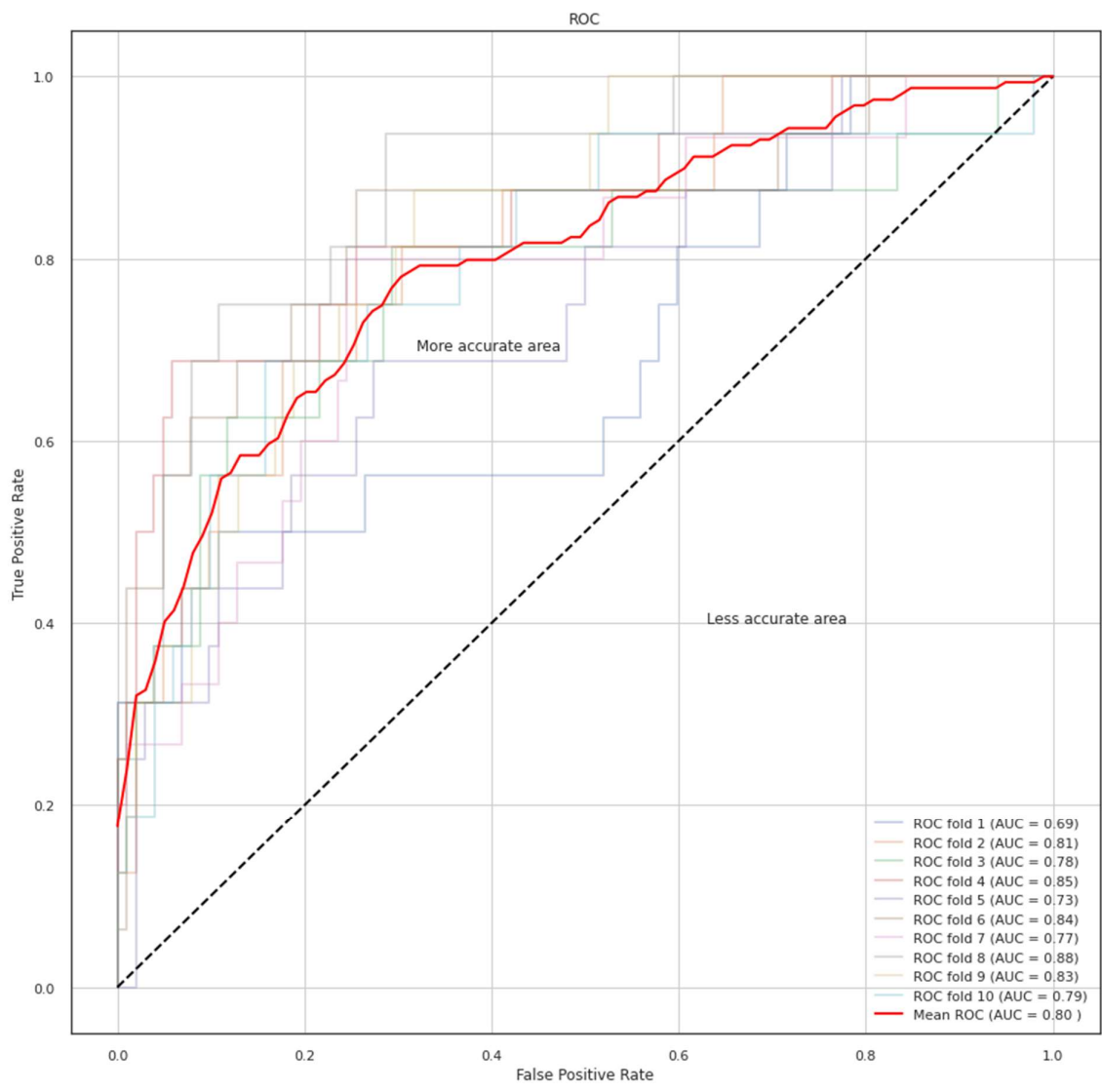


Stratified K-Folds on the raw data results:

Classification Report:

Overall Report:				
	precision	recall	f1-score	support
0.0	0.90	0.97	0.93	1017
1.0	0.64	0.30	0.41	159
accuracy			0.88	1176
macro avg	0.77	0.64	0.67	1176
weighted avg	0.86	0.88	0.86	1176

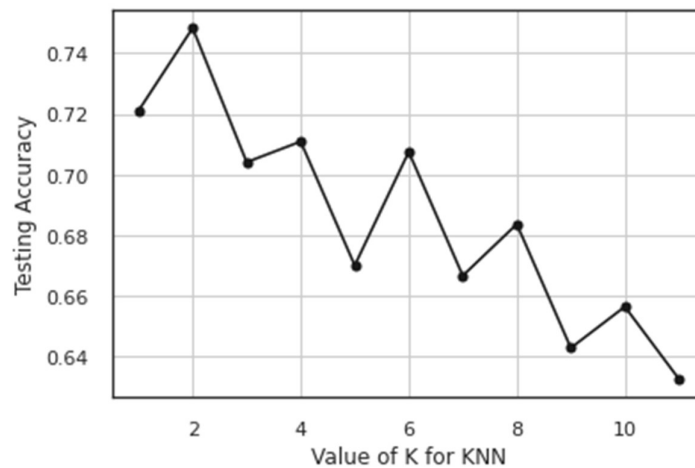
AUC – ROC Curve:



2. K-Nearest Neighbor (KNN):

The supervised learning technique K-nearest neighbors (KNN) is used for both regression and classification. By calculating the distance between the test data and all of the training points, KNN tries to predict the proper class for the test data. Afterward, choose the K number of points that are closest to the test data for the corresponding value.

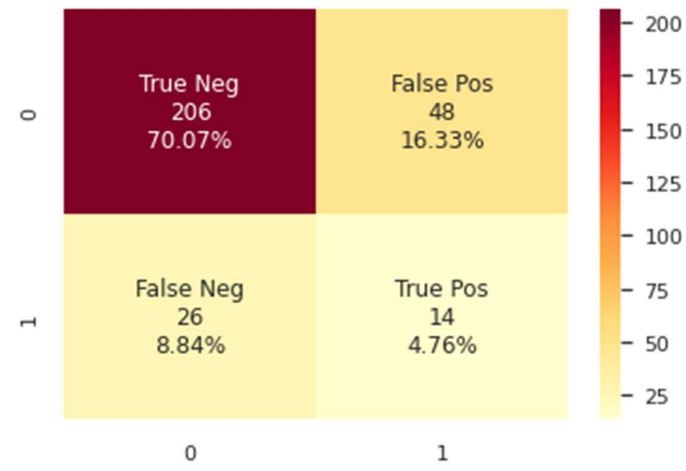
In the oversampled data the best testing accuracy is found at the K value of 2.



Accuracy of The Model:

Accuracy of the model = 0.7482993197278912

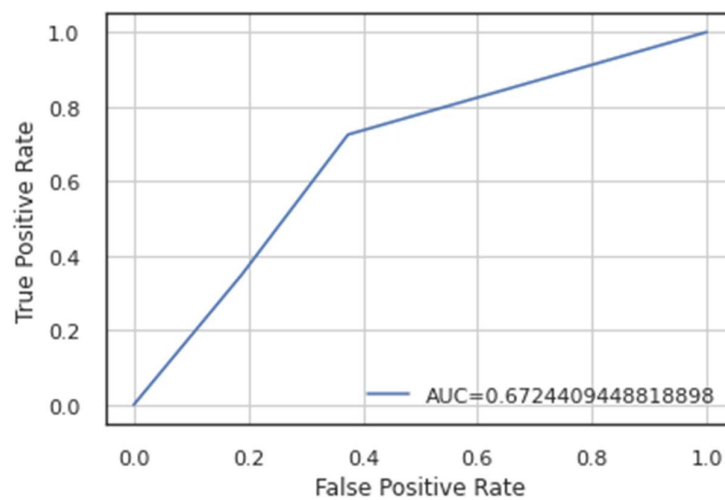
Confusion Matrix:



Classification Report:

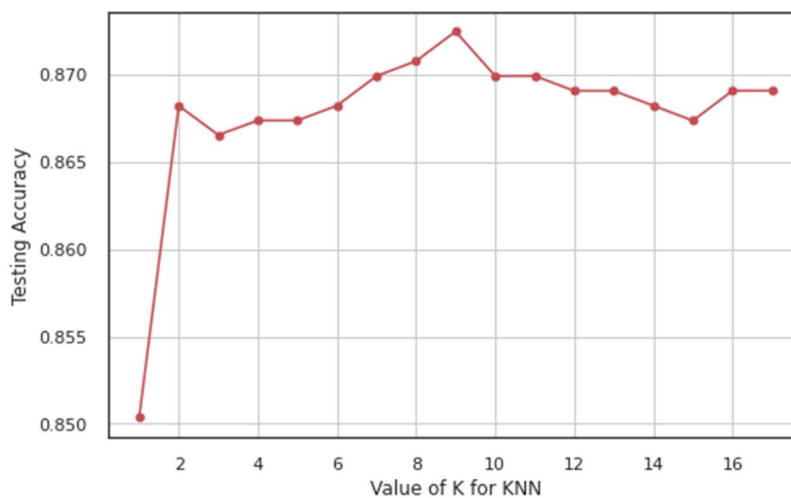
	precision	recall	f1-score	support
0.0	0.89	0.81	0.85	254
1.0	0.23	0.35	0.27	40
accuracy			0.75	294
macro avg	0.56	0.58	0.56	294
weighted avg	0.80	0.75	0.77	294

AUC – ROC Curve:



Stratified K-Folds on the raw data results:

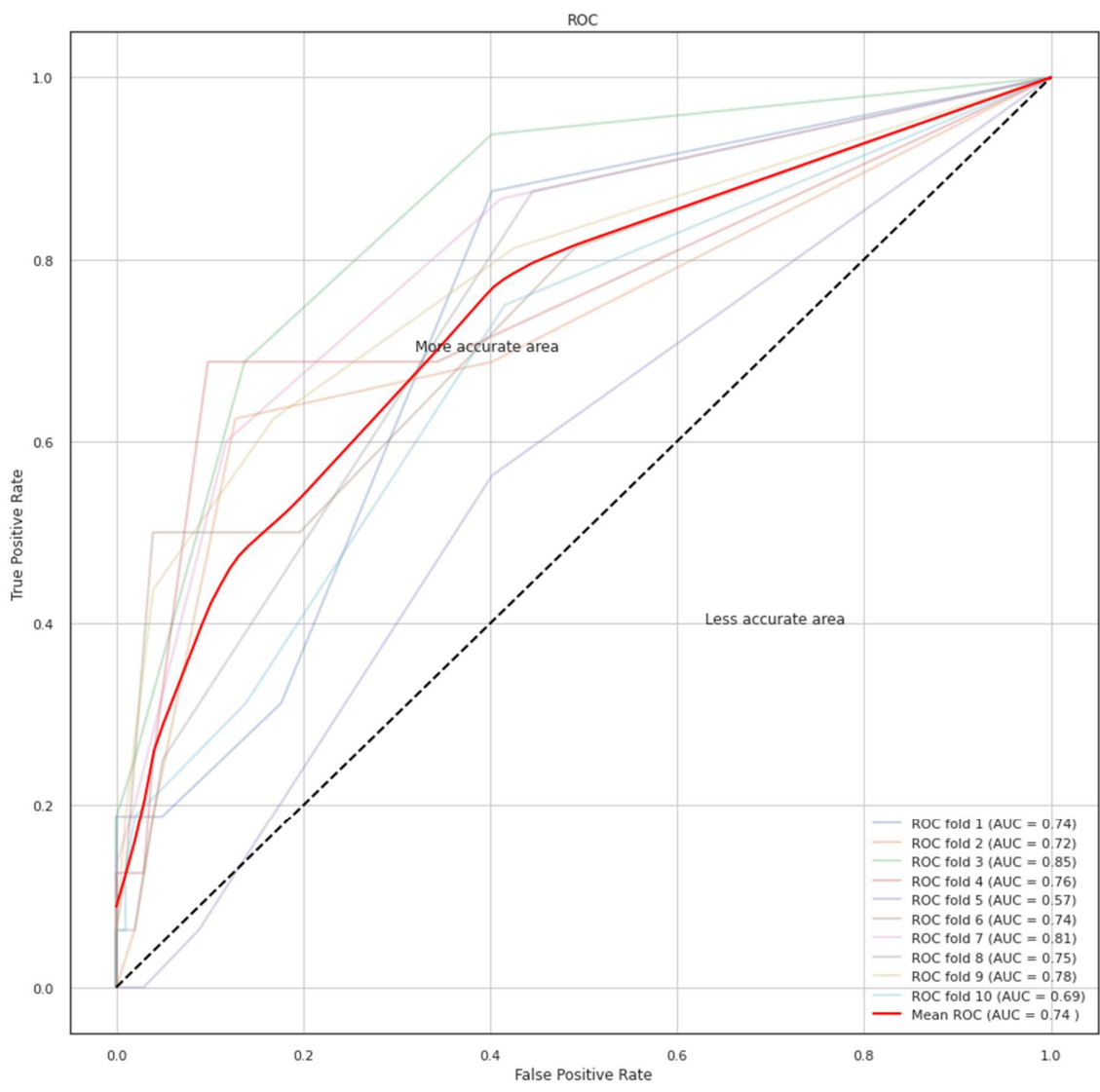
In the original data under stratified K folds the best testing accuracy is found at the K value of 2.



Classification Report:

Overall Report:				
	precision	recall	f1-score	support
0.0	0.87	1.00	0.93	1017
1.0	0.82	0.06	0.11	159
accuracy			0.87	1176
macro avg	0.84	0.53	0.52	1176
weighted avg	0.86	0.87	0.82	1176

AUC – ROC Curve:



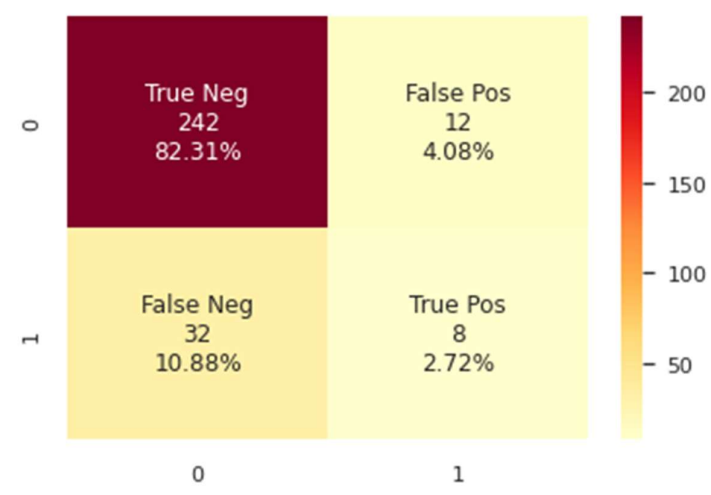
3. Random Forest Classification:

Random forest is a supervised learning algorithm. The “forest” it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Accuracy of The Model:

Accuracy of the model = 0.8503401360544217

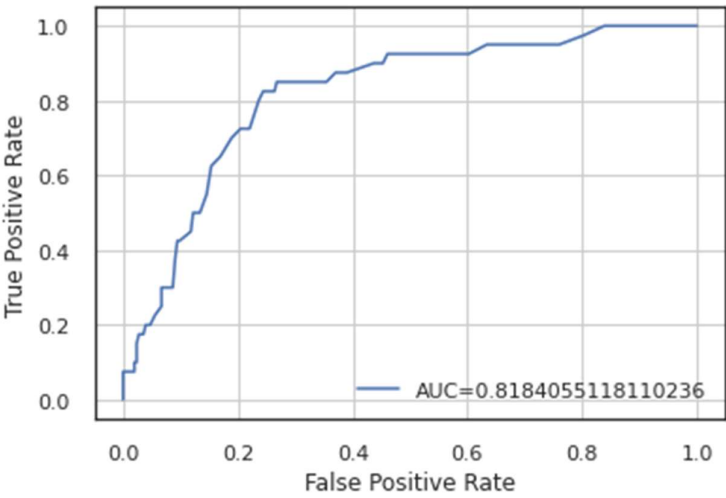
Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0.0	0.88	0.95	0.92	254
1.0	0.40	0.20	0.27	40
accuracy			0.85	294
macro avg	0.64	0.58	0.59	294
weighted avg	0.82	0.85	0.83	294

AUC – ROC Curve:

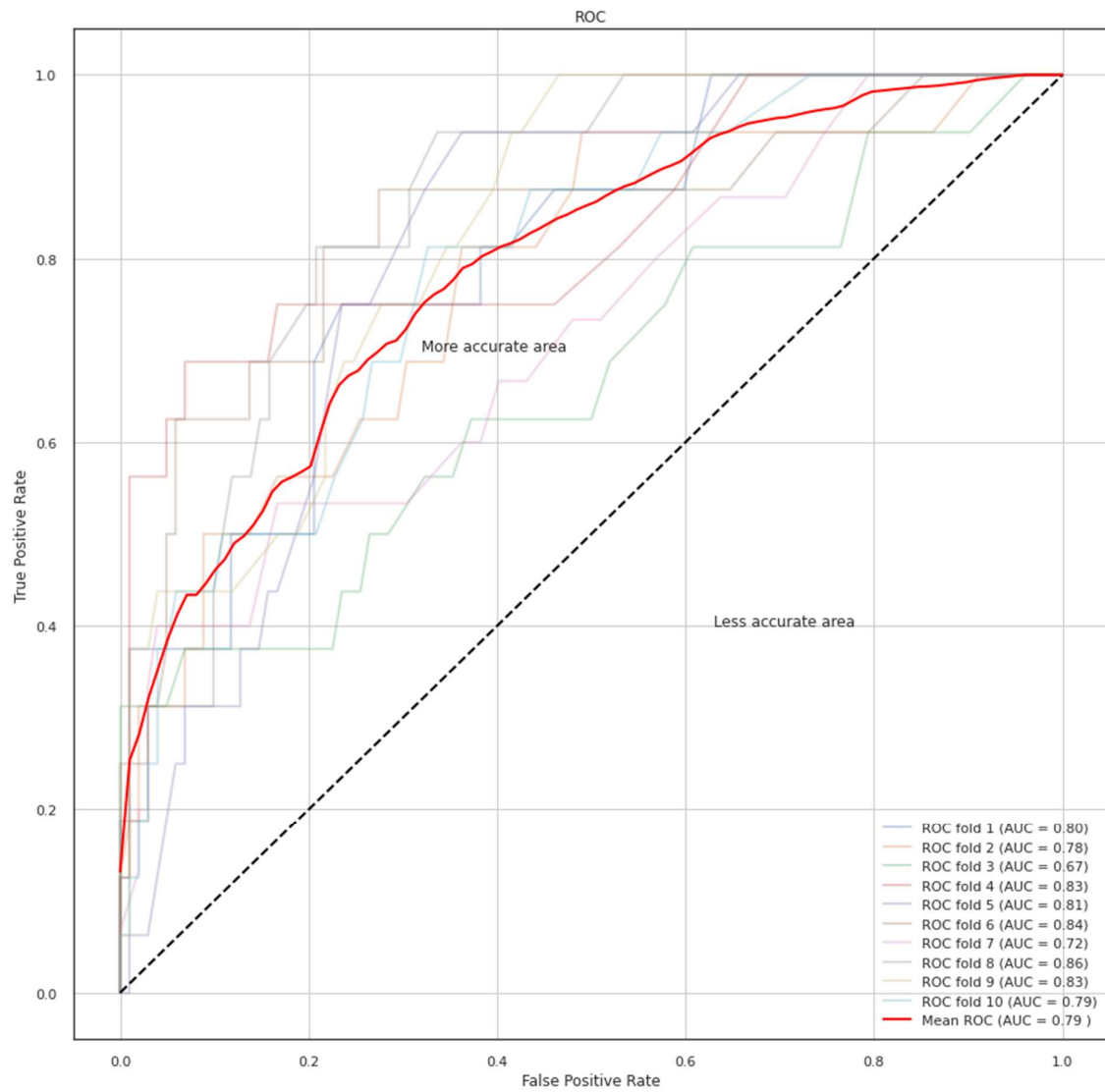


Stratified K-Folds on the raw data results:

Classification Report:

Overall Report:				
	precision	recall	f1-score	support
0.0	0.88	1.00	0.93	1017
1.0	0.81	0.14	0.24	159
accuracy			0.88	1176
macro avg	0.85	0.57	0.59	1176
weighted avg	0.87	0.88	0.84	1176

AUC – ROC Curve:



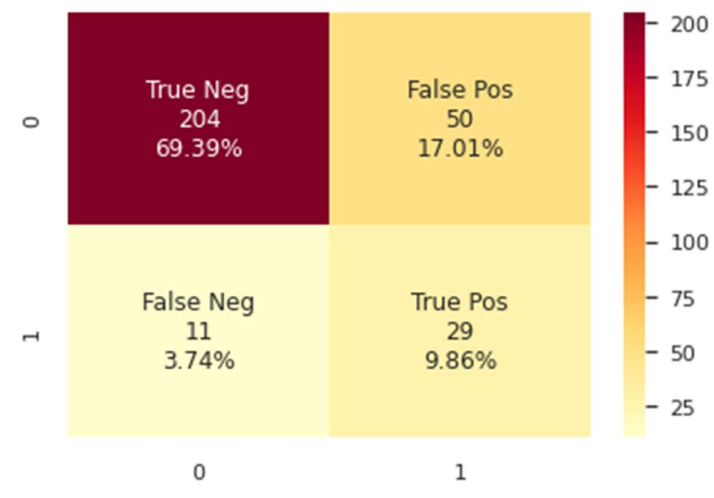
4. Support Vector Machine (SVM):

It is a system of classification. The value of each feature is represented by the value of a specific coordinate in this technique, which plots each data point as a point in n-dimensional space (where n is the number of characteristics you have). For classification and regression issues, SVM is a linear model. It works well for many real-world issues and can solve both linear and non-linear problems. The SVM concept is straightforward: The method divides the data into classes by drawing a line or a hyperplane. the prerequisite that must be met for a training instance to qualify as a support vector.

Accuracy of The Model:

```
Accuracy of the model = 0.7925170068027211
```

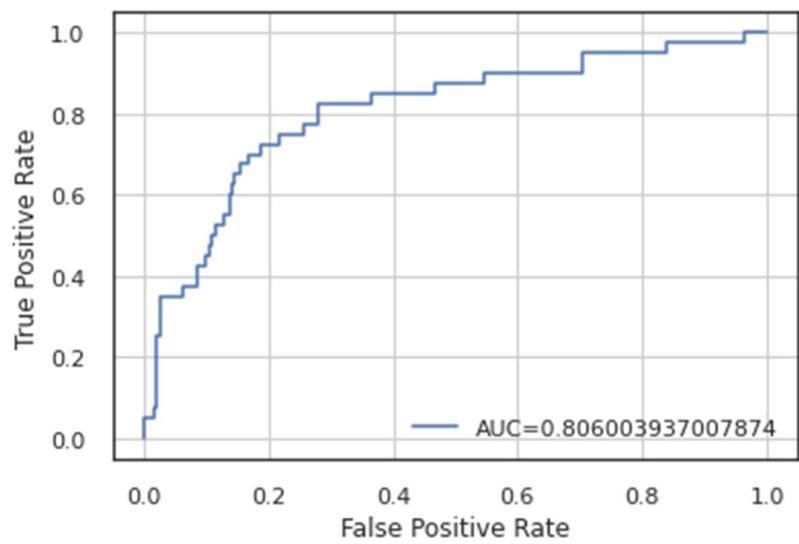
Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0.0	0.95	0.80	0.87	254
1.0	0.37	0.72	0.49	40
accuracy			0.79	294
macro avg	0.66	0.76	0.68	294
weighted avg	0.87	0.79	0.82	294

AUC – ROC Curve:

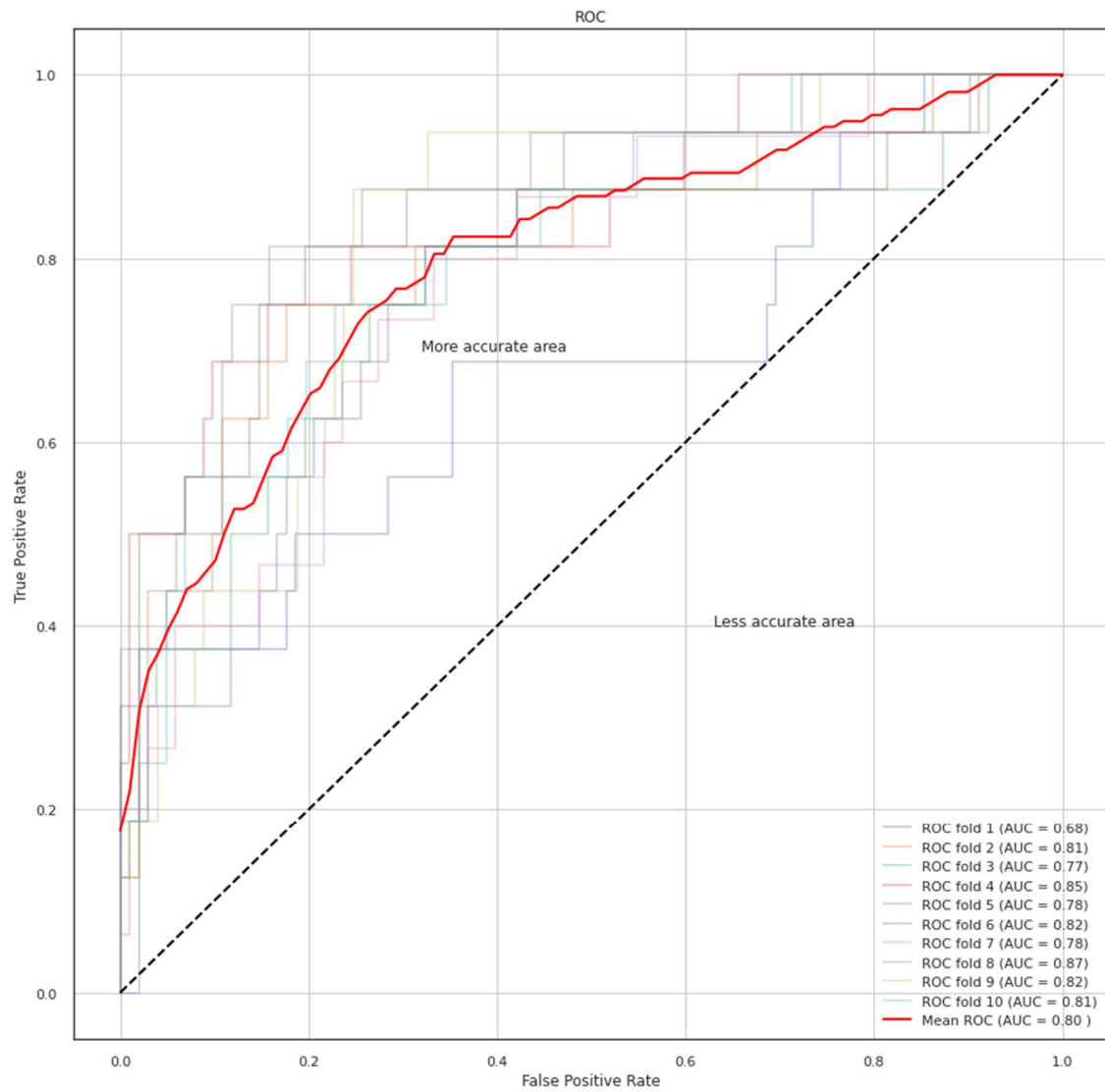


Stratified K-Folds on the raw data results:

Classification Report:

Overall Report:					
	precision	recall	f1-score	support	
0.0	0.87	1.00	0.93	1017	
1.0	0.76	0.08	0.15	159	
accuracy			0.87	1176	
macro avg	0.82	0.54	0.54	1176	
weighted avg	0.86	0.87	0.83	1176	

AUC – ROC Curve:



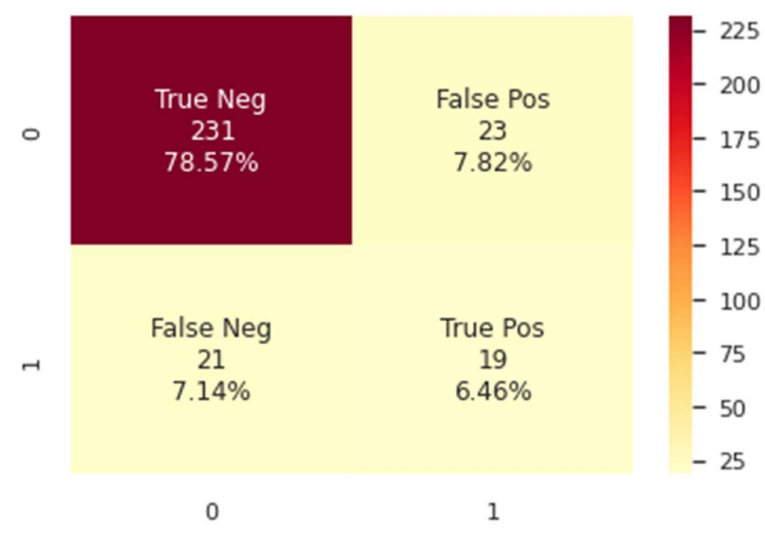
5. Adaptive Boosting (AdaBoost):

AdaBoost, also known as Adaptive Boosting, is a machine learning method used in an ensemble setting. Decision trees with one level, or Decision trees with only one split, are the most popular algorithm used with AdaBoost. In order to boost the effectiveness of binary classifiers, these trees are also known as decision stumps. AdaBoost uses an iterative process to improve poor classifiers by learning from their errors.

Accuracy of The Model:

```
Accuracy of the model = 0.8503401360544217
```

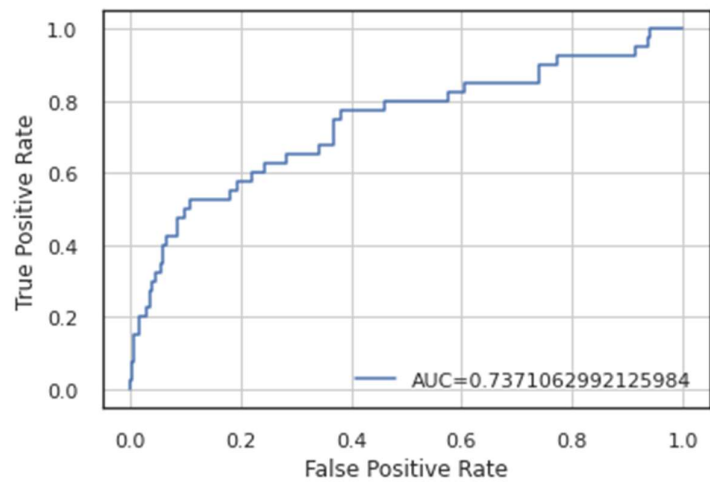
Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0.0	0.92	0.91	0.91	254
1.0	0.45	0.47	0.46	40
accuracy			0.85	294
macro avg	0.68	0.69	0.69	294
weighted avg	0.85	0.85	0.85	294

AUC – ROC Curve:

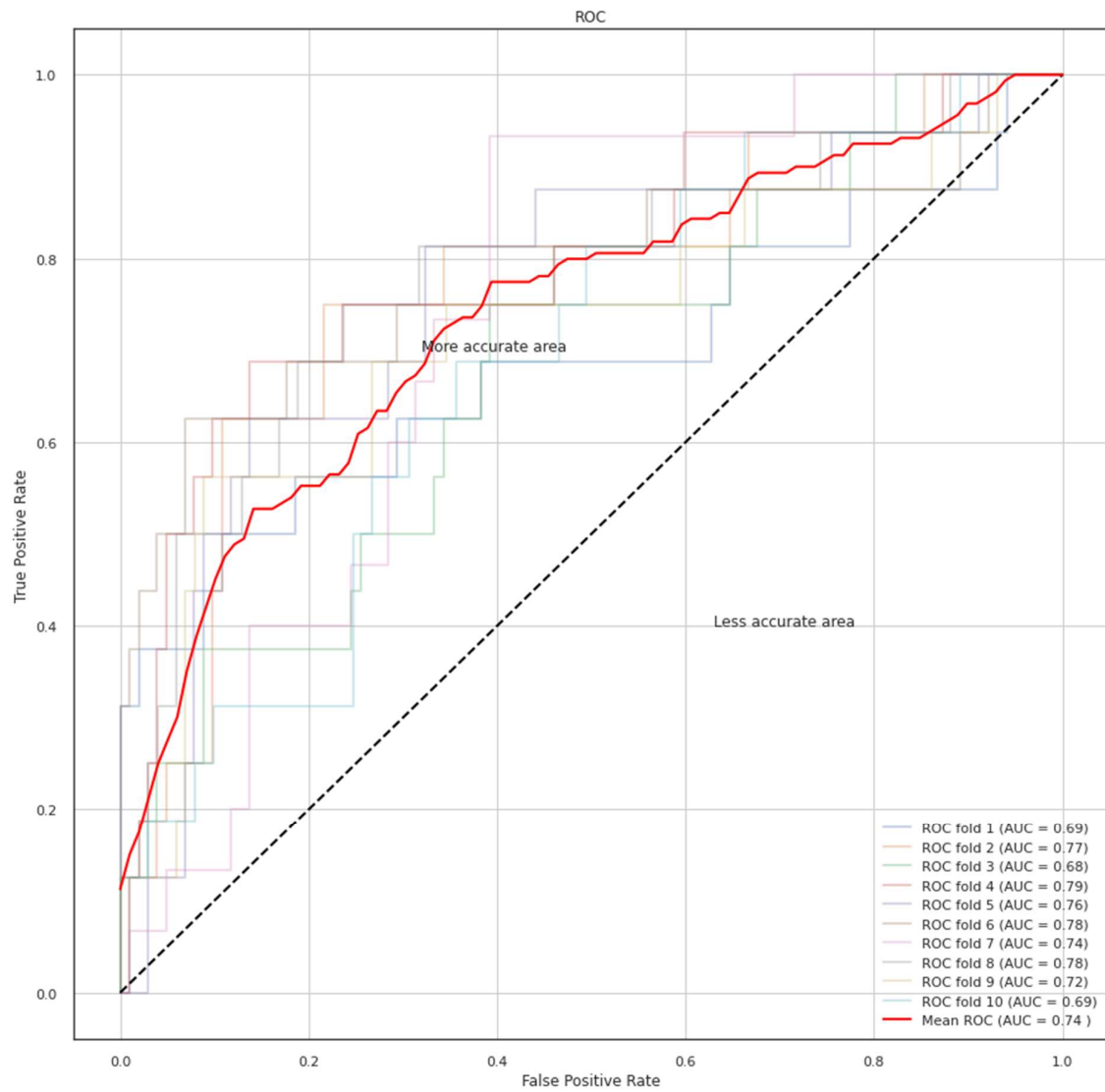


Stratified K-Folds on the raw data results:

Classification Report:

Overall Report:					
	precision	recall	f1-score	support	
0.0	0.89	0.94	0.92	1017	
1.0	0.43	0.29	0.34	159	
accuracy			0.85	1176	
macro avg	0.66	0.61	0.63	1176	
weighted avg	0.83	0.85	0.84	1176	

AUC – ROC Curve:



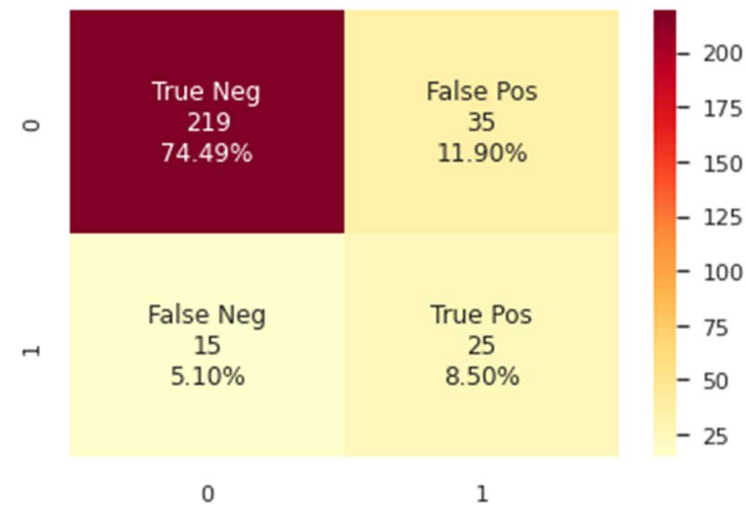
6. Gradient Boosting Classifier:

Before generalizing, the Gradient Boosted Model creates a prediction model made up of a group of decision trees, each of which is a "weak learner," as was the case with Random Forest. Gradient Boosting uses the term "Gradient" to describe the existence of two or more derivatives of the same function. Gradient Boosting is an iterative functional gradient algorithm, meaning that it selects a function that points in the direction of the negative gradient in order to minimize a loss function. In contrast to Random Forest's "bagging" method, it employs the "boosted" machine learning technique. The categorization model employs it. The GBM's ability to build its trees one at a time sets it apart from other tree-building methods. It frequently offers unbeatable predictive accuracy, and it has lots of flexibility—it can optimize on many loss functions and offers a number of hyper parameter tuning options that make the function fit very adaptable.

Accuracy of The Model:

```
Accuracy of the model = 0.8299319727891157
```

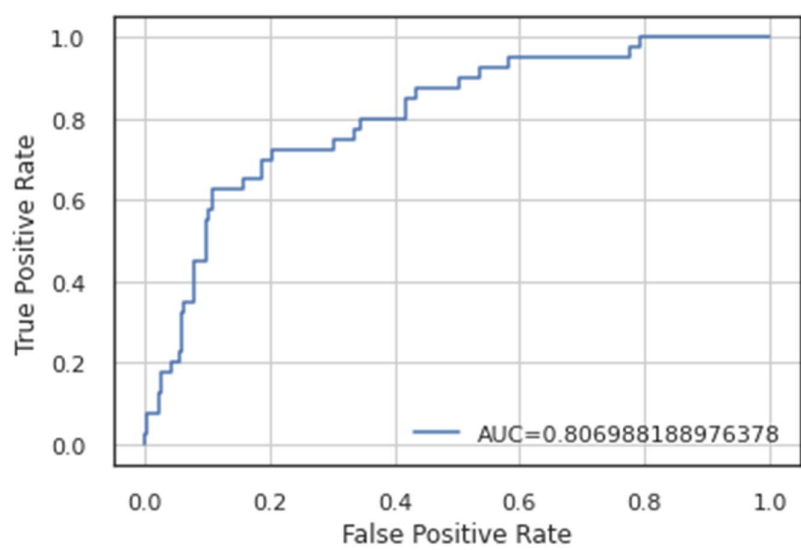
Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
0.0	0.94	0.86	0.90	254
1.0	0.42	0.62	0.50	40
accuracy			0.83	294
macro avg	0.68	0.74	0.70	294
weighted avg	0.87	0.83	0.84	294

AUC – ROC Curve:

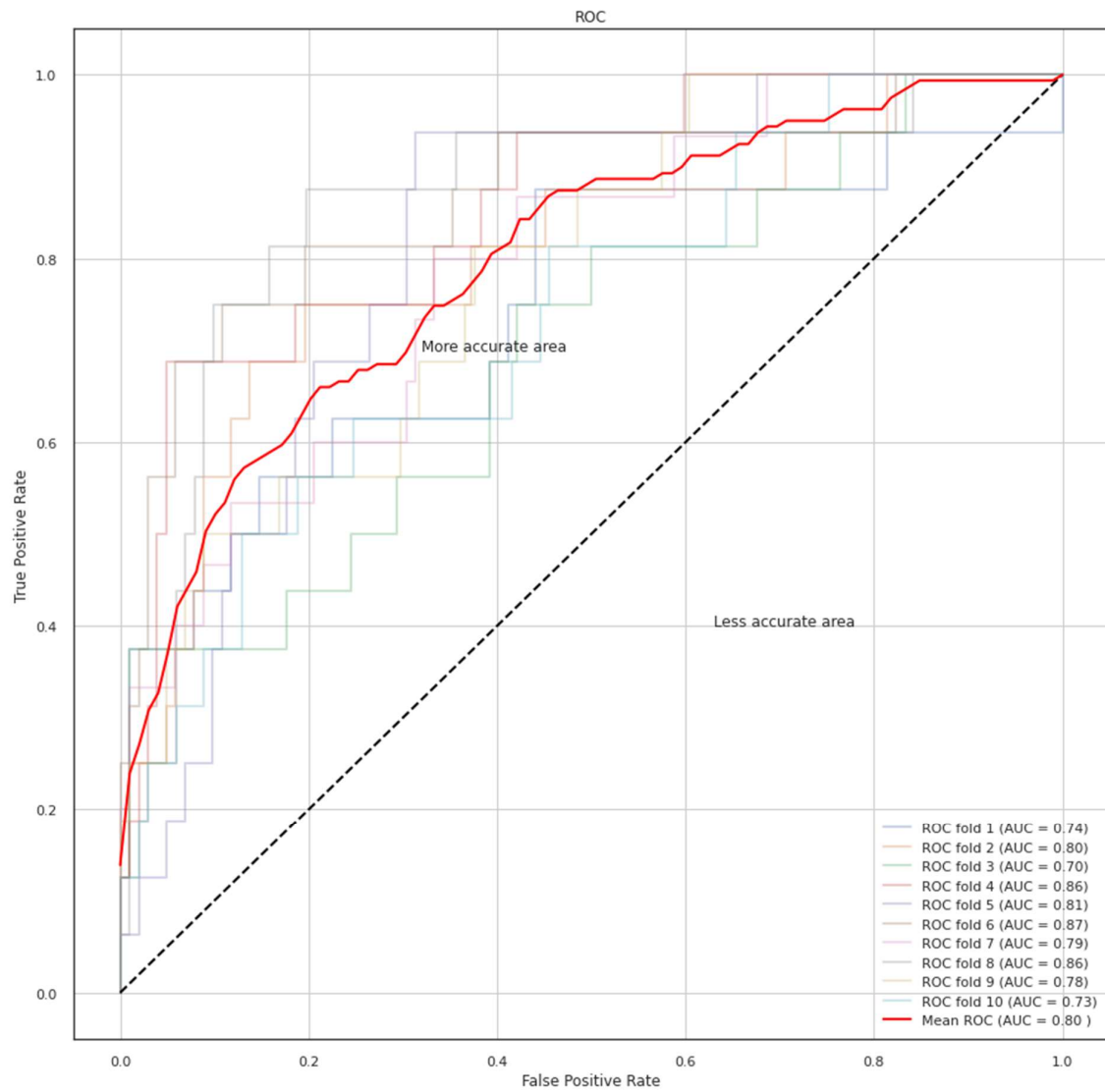


Stratified K-Folds on the raw data results:

Classification Report:

Overall Report:				
	precision	recall	f1-score	support
0.0	0.89	0.98	0.93	1017
1.0	0.63	0.21	0.32	159
accuracy			0.88	1176
macro avg	0.76	0.60	0.63	1176
weighted avg	0.85	0.88	0.85	1176

AUC – ROC Curve:



7. Voting Ensemble:

A voting classifier is a type of machine learning estimator that develops a number of base models or estimators and makes predictions based on averaging their results. Voting for each estimator output can be integrated with the aggregating criteria. There are two categories of voting criteria:

Hard voting: Voting is based on the output class that is anticipated. The class with the highest majority of votes, or the class that had the highest likelihood of being predicted by each of the classifiers, is the predicted output class in hard voting. In this case, the majority anticipated A as the output when three classifiers (A, B, and C) predicted the output class. Therefore, the final prediction will be A.

Soft Voting: Voting is based on the output class's anticipated likelihood. In soft voting, the forecast made for each output class is based on the average probability assigned to that class. Assume that given some input, the prediction probabilities for classes A and B are (0.30, 0.47, and 0.53) and (0.20, 0.32, 0.40). As a result, class A's average is 0.4333, while class B's average is 0.3067. As a result, class A is the winner because it had the highest probability as averaged by all classifiers.

For Oversampled Data with Test-Train-Split:

We evaluate the accuracy of the six models using the voting ensemble.

```
LogisticRegression : 0.7959183673469388
KNeighbours : 0.7482993197278912
RandomForest : 0.8503401360544217
SVM : 0.7925170068027211
AdaBoost : 0.8503401360544217
GradientBoosting : 0.8299319727891157
```

Soft Voting:

	precision	recall	f1-score	support
0.0	0.91	0.91	0.91	254
1.0	0.44	0.45	0.44	40
accuracy			0.85	294
macro avg	0.68	0.68	0.68	294
weighted avg	0.85	0.85	0.85	294

Hard Voting:

	precision	recall	f1-score	support
0.0	0.91	0.93	0.92	254
1.0	0.49	0.45	0.47	40
accuracy			0.86	294
macro avg	0.70	0.69	0.69	294
weighted avg	0.86	0.86	0.86	294

For Original Data with Stratified K-Folds:

We evaluate the accuracy of the six models using the voting ensemble.

```
LogisticRegression : 0.8547008547008547
KNeighbours : 0.8632478632478633
RandomForest : 0.8632478632478633
SVM : 0.8888888888888888
AdaBoost : 0.8290598290598291
GradientBoosting : 0.8717948717948718
```

Soft Voting:

	precision	recall	f1-score	support
0.0	0.92	1.00	0.95	254
1.0	0.94	0.42	0.59	40
accuracy			0.92	294
macro avg	0.93	0.71	0.77	294
weighted avg	0.92	0.92	0.90	294

Hard Voting:

	precision	recall	f1-score	support
0.0	0.91	1.00	0.95	254
1.0	0.93	0.35	0.51	40
accuracy			0.91	294
macro avg	0.92	0.67	0.73	294
weighted avg	0.91	0.91	0.89	294

Discussion:

The first task for our project was to understand and clean the dataset. The dataset we used is quite small and not entirely clean for ready use. Our dataset had quite a number of missing values and we had to find corresponding ways to remove them. First, we inspected the dataset to see the number of instances and features. Then we found out the summed values of the missing data entries. After that we used the value count function to find the imbalance in our dataset. We have found our dataset is imbalanced. Now we had to process the data and remove any missing values. The approach we took to impute the missing values is to use the KNN Imputer.

For missing value imputation there are various methods such as mean, KNN, MICE etc. In our dataset we are dealing with biomedical data. From our understanding we figured that mean values may not be a good replacer. Rather comparing the data with the nearest neighbors and then replacing according to the trend should be a much more feasible approach. So first, we standardized the data and then we used the KNN Imputer and after doing this all the missing values were replaced with appropriate values.

Then we used the 'BorderlineSMOTE' to oversample. We did this because our dataset was imbalanced and a lot of the times classifiers give inaccurate results for imbalanced classes. To combat this imbalance problem one approach is to resample the data by either oversampling or undersampling or a combination of both. As our dataset is not that big in size, we crossed out anything related to undersampling. After careful consideration we decided on using oversampling and more specifically 'BorderlineSMOTE' oversampling. SMOTE is a popular oversampling technique and a variation of it is the borderlineSMOTE that we have used. We used this one because in this oversampling the minority class issue is solved where minority classes with many majority class neighbors are marked as noise when oversampling. BorderlineSMOTE doesn't bring huge changes rather a very small change in the results.

Then we did a principal component analysis for both the main classes and oversampled classes. The component gives us an easier idea on the class density and distribution.

After that we also found out the correlation heatmap. This was done to see the relations between different features. The heatmap gave us an idea on the overall relation among the features and also if there were any features we could ignore or remove from our tests.

Now we did two things for our model testing. Primarily we used a 'Train-Test Split' to split our oversampled data with a testing size of 25%. We used this on all the models we tested. Then again, we use 'Stratified K-Folds' to split our main data without any oversampling with a 10-fold operation.

Now with all these done we moved on to our model testing. For every model we evaluated the accuracy of the model, confusion matrix of the model, classification report of the model and the AUC-ROC score. We did this for both the oversample test-train split data and no sample stratified k-folds data. The results for each step were different, in some cases we had good accuracy but the recall and f-1 score was not satisfactory.

A comparison of accuracy can be seen here:

Oversampled Data (Test-Train Split):

```
LogisticRegression : 0.7959183673469388
KNeighbours : 0.7482993197278912
RandomForest : 0.8503401360544217
SVM : 0.7925170068027211
AdaBoost : 0.8503401360544217
GradientBoosting : 0.8299319727891157
```

Main Data (Stratified K-Folds):

```
LogisticRegression : 0.8547008547008547
KNeighbours : 0.8632478632478633
RandomForest : 0.8632478632478633
SVM : 0.8888888888888888
AdaBoost : 0.8290598290598291
GradientBoosting : 0.8717948717948718
```

Our target here was to maximize the prediction of class '1' instances. So, we tried to focus on improving our recall score. And for that we also employed a special algorithm – 'Voting Ensemble.'

Using the voting ensemble, we compared the results of our models and then using the hard voting and soft voting techniques we tried to improve our overall classification results. And in the classification results we have seen improvements.

In our evaluation we did not rely only on the accuracy of the models as our primary performance metrics. Rather we tried to figure out other performance parameters to more solidify our results and get a better understanding on them. We employed the popular ones such as the confusion matrix, the classification reports, the AOC-RUC scores.

The confusion matrix gave us a grasp on the prediction analysis. We wanted more true positives and so our target was to increase the recall in our classification score also the f1 score. To do that we tried the voting ensemble method. And we got expected results much more improved than the original model generated results. So, it was a good decision for us to employ that.

Future Plans:

Our project has a lot of room for improvement. For our future endeavors we plan on –

1. Employing the g-mean score, sensitivity, specificity and related performance parameters.
2. Test the uses of ‘imblearn’ library to apply Stratified K-Folds with oversampled data.
3. Trying ‘Feature Engineering’ to find more important features among the bunch.
4. Employing other types of correlation heatmaps.
5. Trying a new variety of models.

As our first work on machine learning we couldn’t try everything but in our future plan we will focus and work on improving this understanding of ours.

Conclusion:

In this machine learning project, we tried various methods to train and test our classification problem. We tried to bring in six different classifier models and focused on finding their performance on the basis of different parameters such as the accuracy, the confusion matrix, the classification report, aur-roc score. This was done so that we can accurately understand how our model is working for the targeted classifier. Our data had two divisions among them. First portion is comprised of oversampled data run using a ‘Test-Train Split’ and the second portion is comprised of original data using ‘Stratified K-Folds’ to see which method gives us a better result. In order to improve the classification scores, we have also adopted the ‘Voting Ensemble’ method. This method used the results using our previously tested models and produced a better result for the classification report.

References:

1. <https://www.kaggle.com/datasets/saurabhshahane/in-hospital-mortality-prediction?resource=download>
2. Choi, M.H., Kim, D., Choi, E.J., Jung, Y.J., Choi, Y.J., Cho, J.H. and Jeong, S.H. (2022). Mortality prediction of patients in intensive care units using machine learning algorithms based on electronic health records. *Scientific Reports*, 12(1). doi:10.1038/s41598-022-11226-4.
3. Sadeghi, R., Banerjee, T. and Romine, W. (2018). Early hospital mortality prediction using vital signals. *Smart Health*, [online] 9-10, pp.265–274. doi:10.1016/j.smhl.2018.07.001.
4. Lu, S.-C., Xu, C., Nguyen, C.H., Geng, Y., Pfob, A. and Sidey-Gibbons, C. (2021). Machine Learning-based Short-term Mortality Prediction Models for Cancer Patients Using Electronic Health Record Data: A Systematic Review and Critical Appraisal (Preprint). *JMIR Medical Informatics*. doi:10.2196/33182.
5. Kong, G., Lin, K. and Hu, Y. (2020). Using machine learning methods to predict inhospital mortality of sepsis patients in the ICU. *BMC Medical Informatics and Decision Making*, 20(1). doi:10.1186/s12911-020-01271-2.
6. Subudhi, S., Verma, A., Patel, A.B., Hardin, C.C., Khandekar, M.J., Lee, H., McEvoy, D., Stylianopoulos, T., Munn, L.L., Dutta, S. and Jain, R.K. (2021). Comparing machine learning algorithms for predicting ICU admission and mortality in COVID-19. *npj Digital Medicine*, [online] 4(1). doi:10.1038/s41746-021-00456-x.

7. Anand, R.S., Stey, P., Jain, S., Biron, D.R., Bhatt, H., Monteiro, K., Feller, E., Ranney, M.L., Sarkar, I.N. and Chen, E.S. (2018). Predicting Mortality in Diabetic ICU Patients Using Machine Learning and Severity Indices. AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science, [online] 2017, pp.310–319. Available at:
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5961793/>.
8. Thorsen-Meyer, H.-C., Nielsen, A.B., Nielsen, A.P., Kaas-Hansen, B.S., Toft, P., Schierbeck, J., Strøm, T., Chmura, P.J., Heimann, M., Dybdahl, L., Spangsege, L., Hulsén, P., Belling, K., Brunak, S. and Perner, A. (2020). Dynamic and explainable machine learning prediction of mortality in patients in the intensive care unit: a retrospective study of high-frequency data in electronic patient records. *The Lancet Digital Health*, 2(4), pp.e179–e191. doi:10.1016/s2589-7500(20)30018-2.