Diabetes Prediction

Group Members:

Subhajit Ghatak
Asansol Engineering College
211080571010015

Saurabh Thakur
Asansol Engineering College
211080571010011

Ashish Dungdung
Asansol Engineering College
211080571010020

Charanjit Singh
Asansol Engineering College
211080571010034

Smriti Sengupta
Asansol Engineering College
211080571010058

Table of Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring, and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

……………………………..
Subhajit Ghatak

……………………………..
Saurabh Thakur

……………………………..
Ashish Dungdung

……………………………..
Charanjit Singh

……………………………..
Smriti Sengupta

Project Objective

In this project we have a shortened ‘Predict Diabetes’ Data set from Kaggle. In this data set, the target attribute is the outcome which is binary. So, in this project we need to do binary classification based on the attributes present in our data set and predict whether a patient have diabetes or not.

Our objective in this project is to study the given data set of ‘Predict Diabetes’. We might need to pre-process the given data set if we need to. Then, we would train 4 models viz. ‘KNN classifier model’, ‘Naive Bayes classifier model’, ‘Decision Tree Classifier’ and ‘Linear Regression classifier model’. After training the aforementioned models, we will need to find out the score, classification report, plot the Receiver Operating Characteristic graph and find out the Area Under Curve(AUC) for each of the models trained. Our next step would be to use the trained models to predict the outcomes using the given test data set and compare the outcome of each model. We would then choose the best model based on the accuracy score and classification report.

Our methodology for solving the problems in the given project is described below:

- Load the required data set.
- Study the data set.
- Describe the data set.
- Visualize the data set.
- Find out if the data set needs to be pre-processed.

1. It will be determined on the basis of whether the data set has null values of outliers or any such discrepancy that might affect the output of the models to be trained.

- If the data set is required to be pre-processed, take the necessary steps to pre-process the data.
- Find out the principal attributes for training.
- Split the given data set for training the testing purpose.
- Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test data set.
- Choose the best model among the 4 trained models bases on the accuracy and classification reports.

Project Scope

The broad scope of ‘Predict Diabetes’ project is given below:

- The given data set has attributes based on which the diabetes will be predicted.
- It is a useful project as the Classifier models can be used to quickly determine the outcome of diabetes of large data sets.
- Various hospital institution can use these models and modify them according to their needs to use in their diabetes prediction. This will reduce the manual labor and time spent on determining whether the patient has diabetes or not.
- Patients who are unaware of them having diabetes or not, can use these trained models to check for possibility. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- The data set given to us is a shortened form of the original data set from Kaggle. So, the results might have some mismatch with the real-world applications. But that can be avoided if the models are trained accordingly.

Data Description

Source of the data: Kaggle, the given data set is a shortened version of the original data set in Kaggle.

Data Description: The given train data set has 768 rows and 7 columns.

Columns	**Attribute Name**	**Type**	**Description**	**Target Attribute**
Glucose	glucose	non categorical	Glucose level in blood.	No
Blood Pressure	Bp	non categorical	Blood pressure of the patient.	No
Insulin Level	insulin	non categorical	Insulin level in blood.	No
BMI	bmi	non categorical	Body mass index of the patient.	No
Diabetes Pedigree Function	pedigree	non categorical	To express diabetes percentage.	No
Age	Age	non categorical	Age of the patient.	No
Outcome	outcome	categorical	To express the final result Yes or No.	Yes

Table 1: data description

The following table shows the 5 number statistics of the given data set:

	glucose	**bp**	**insulin**	**bmi**	**pedigree**	**age**
Mean	120.89	69.07	79.79	32.00	0.47	33.24
Standard deviation	31.97	19.35	115.24	7.88	0.33	11.76
Minimum	0.00	0.00	0.00	0.00	0.07	21.00
25%	99.00	62.00	0.00	27.30	0.24	24.00
50%	117.00	72.00	30.50	32.00	0.37	29.00
75%	140.25	80.00	127.25	36.00	0.62	41.00
Maximum	199.00	122.00	846.00	67.00	2.42	81.00

Table 2: 6 number statistics of the given dataset

Now we will pre-process the data. The methodology followed is given below:

- Checking for null values
  - If null values are present, we will fill them or drop the row containing the null value based on the data set.
- Checking for outliers.
  - If outliers are present, they will either be removed or replaced by following a suitable method depending on the data set.

Data Pre-Processing

As the given data set had Categorical and Non-categorical data mixed, we converted the categorical data into non-categorical data accordingly. We converted the binary categories into 0 and 1. We converted the other categorical attributes into suitable numerical values.

The following table shows the conversion record:

Non-Numeric to Numeric Change table		
Column	Initial Value	Replaced Value
Outcome	No	0
	Yes	1

Table 3: Categorical to Numerical change in values

We searched for null values in out data set and formed the following table:

Column Name	**Count of Null Value**
glucose	0
bp	2
insulin	0
bmi	2
pedigree	0
age	0
outcome	0

table 4: Count of Null values

To visualize the null values, we made a heat map plot using seaborn library function heat map. The heat map plot is given below:

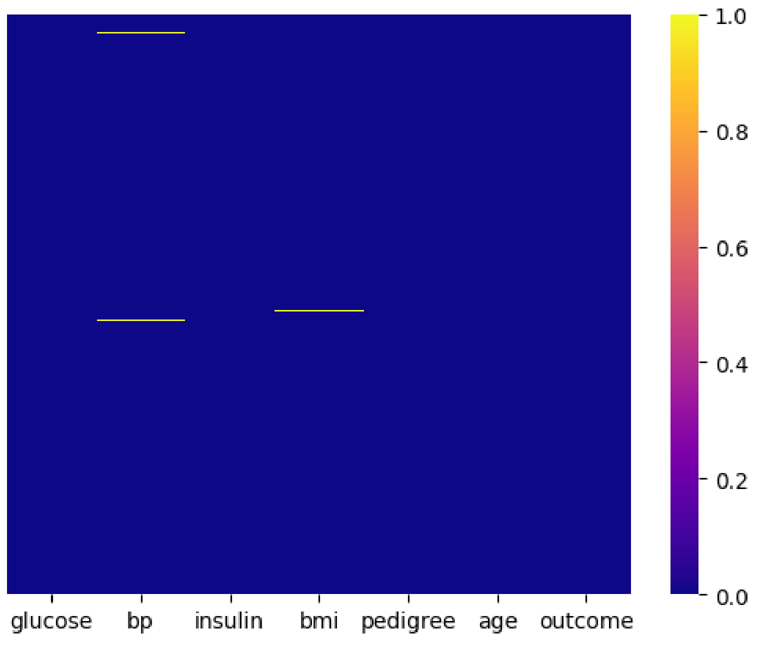


Fig 1: Heatmap of the given Data

The heat map shows that the data set has null values

To remove the null values, we had the following methodology:

filling Null values in ‘bp’ and ‘bmi’ with mean (as started in the data set description from source)

After removing the null values, the following heat map was obtained:

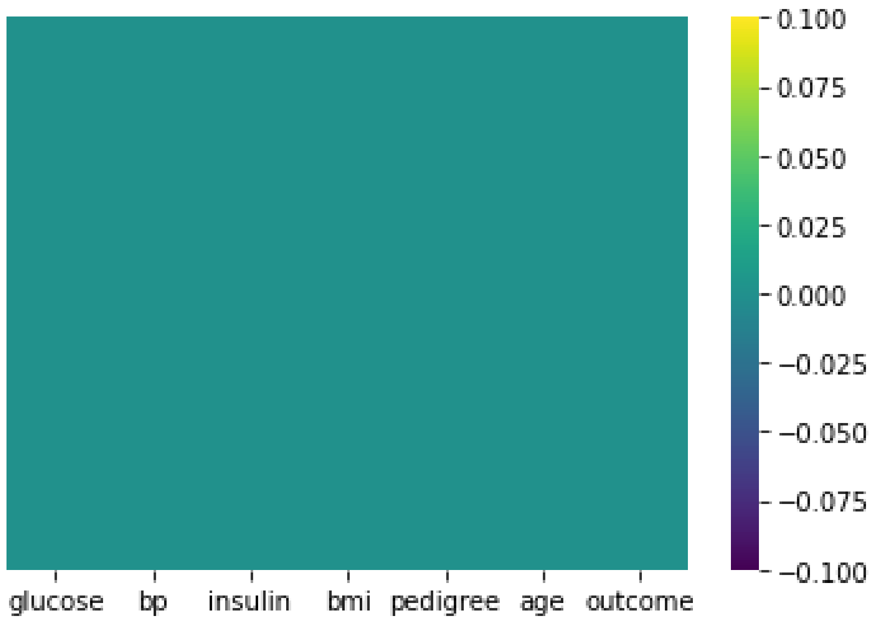


Fig 2: Heatmap implying absence of null values

Now we have successfully handled Null values and converted non-numeric values to Numeric values. We didn't drop the rows with null values as we have a small data set (only 100 entries).

So, we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'Outcome' column in the data set.

The following table gives the correlation value of each attribute with our target attribute i.e. 'Outcome'.

Columns	**Correlation value**
glucose	0.466581
bp	0.063315
insulin	0.130548
bmi	0.292956
pedigree	0.173844
age	0.238356

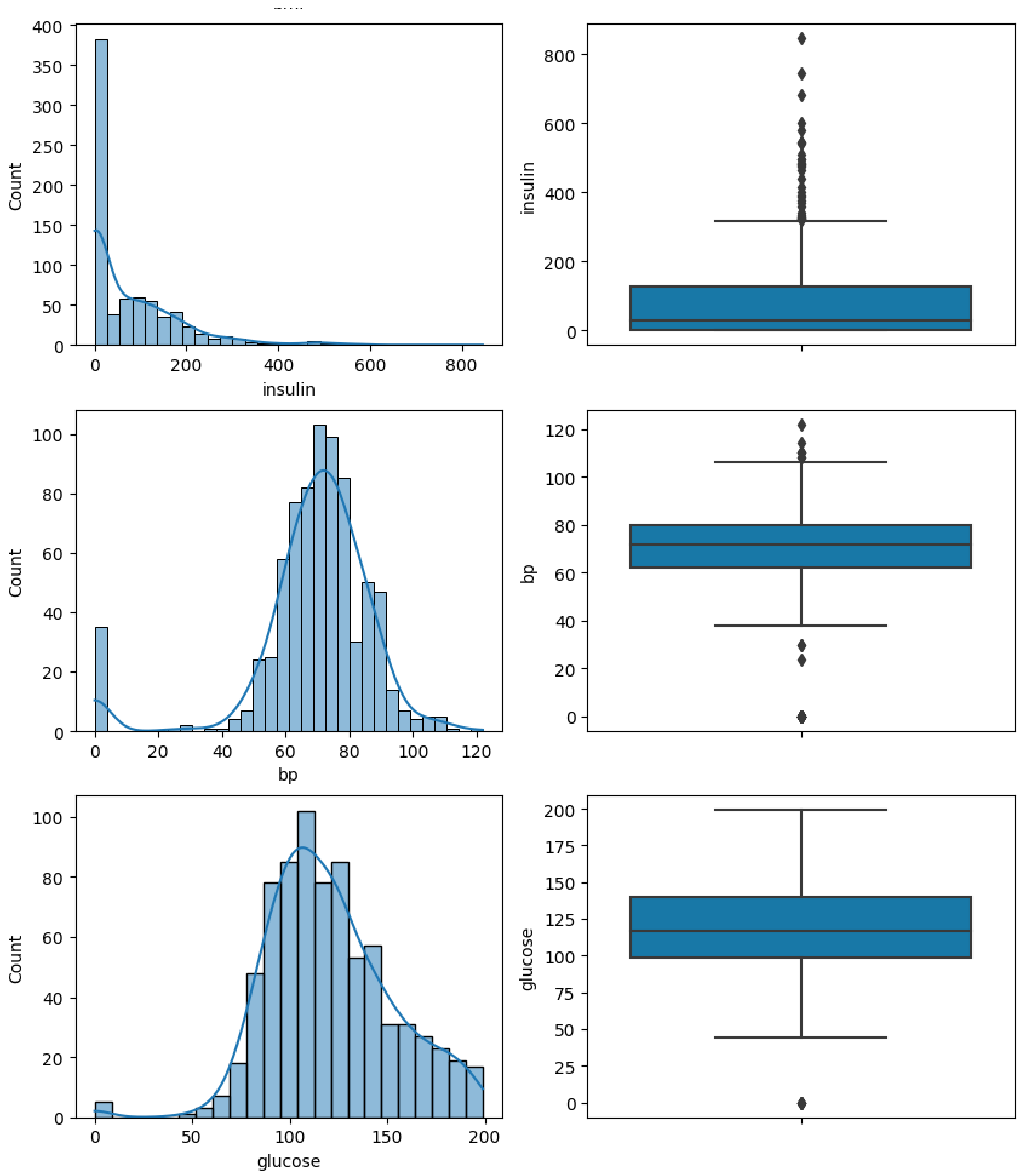
Table 5: Correlation values with target attribute

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

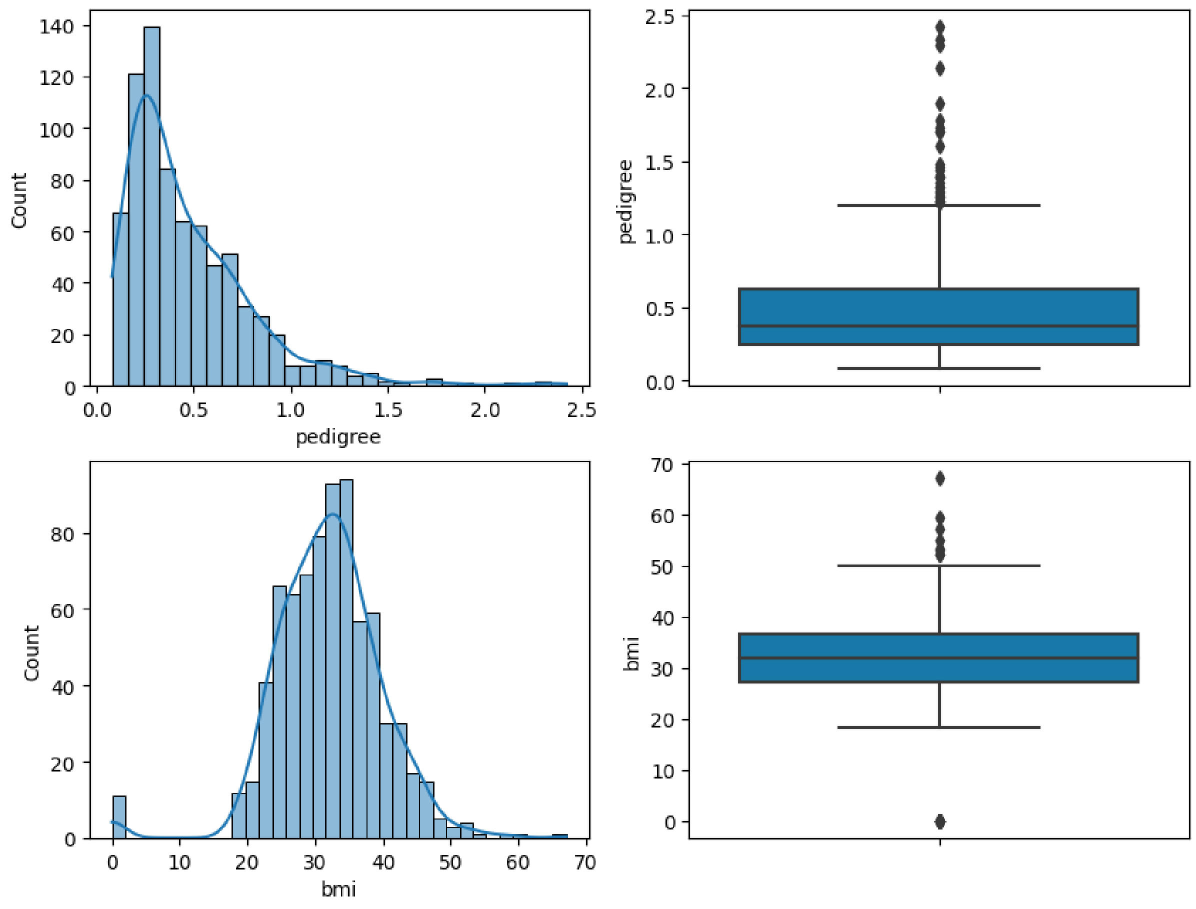
Most common causes of outlier on a data set:

- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

We plot distribution graph and box plot to visualize the outliers. The plots are given in next page:



Distribution plot and boxplot of insulin bp glucose showing possible outliers



Distribution plot and boxplot of pedigree and bmi showing possible outliers

Fig 3: Distribution plot and boxplot of 5 attributes
pedigree, bmi, insulin, bp, glucose

From the distribution plots and boxplots, we can see that there are possible outliers in the columns ‘pedigree’, ‘bmi’, ‘insulin’, ‘bp’, ‘glucose’.

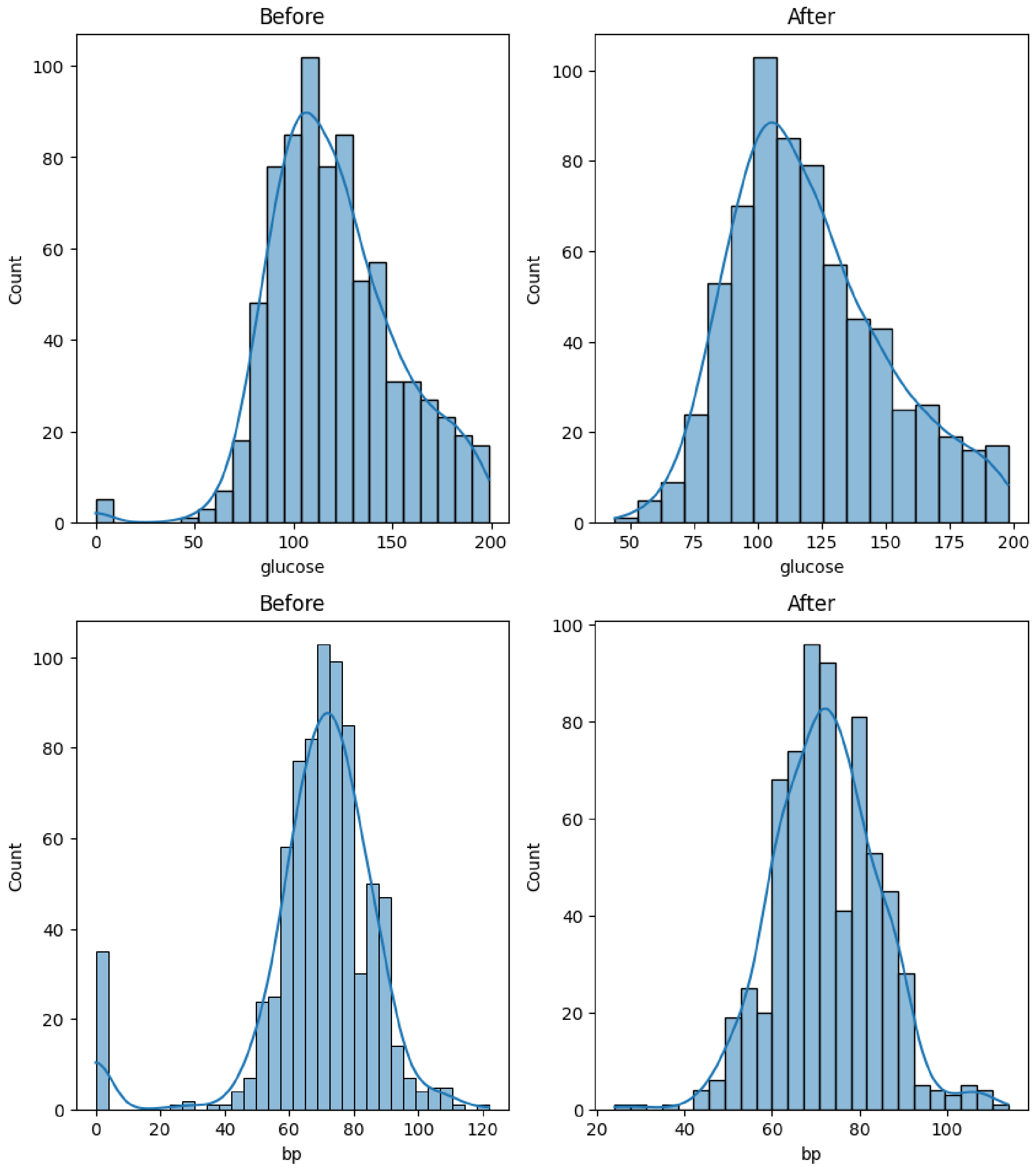
We have not checked for outliers in the column ‘outcome’ as it has binary values and also initially non-numeric values.

Now, we had to handle the outliers. We handled the outliers using parametric method. We used z-score to handle the outliers. In this method we’re calculating z-score of each data points. The formula is Z Score = $(x - \overline{x})/\sigma$

Here, x = random data point
$\overline{x}$ = mean of the column in which the data point belongs
σ = standard deviation

After using this formula for every data point, we get a z-score for every data point. Now we’re eliminating those data points that having z-score greater than 2.5.

After handling the data, we re-plot the same distribution and box plot but this time using the modified data. We obtain the following plots:



Before After

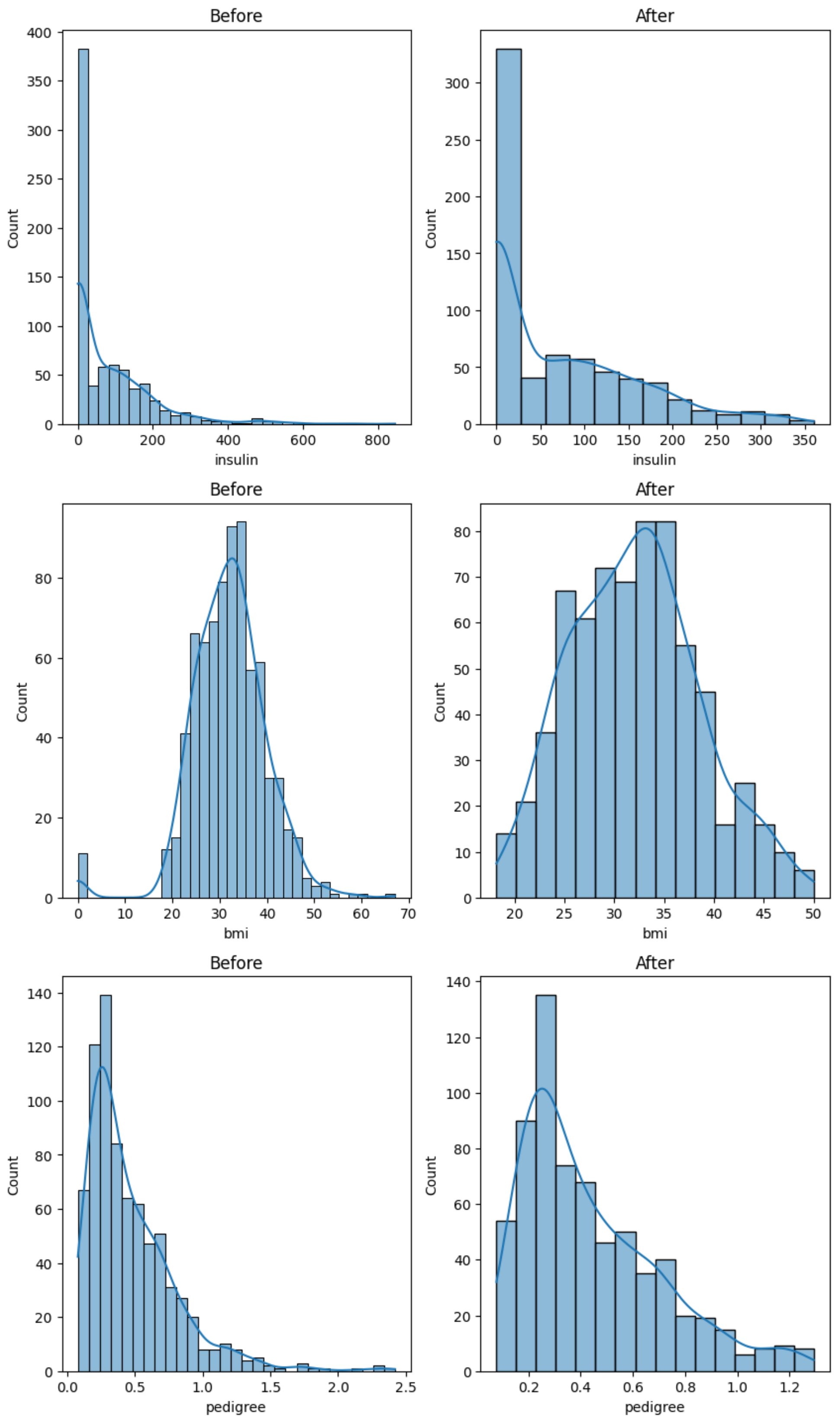
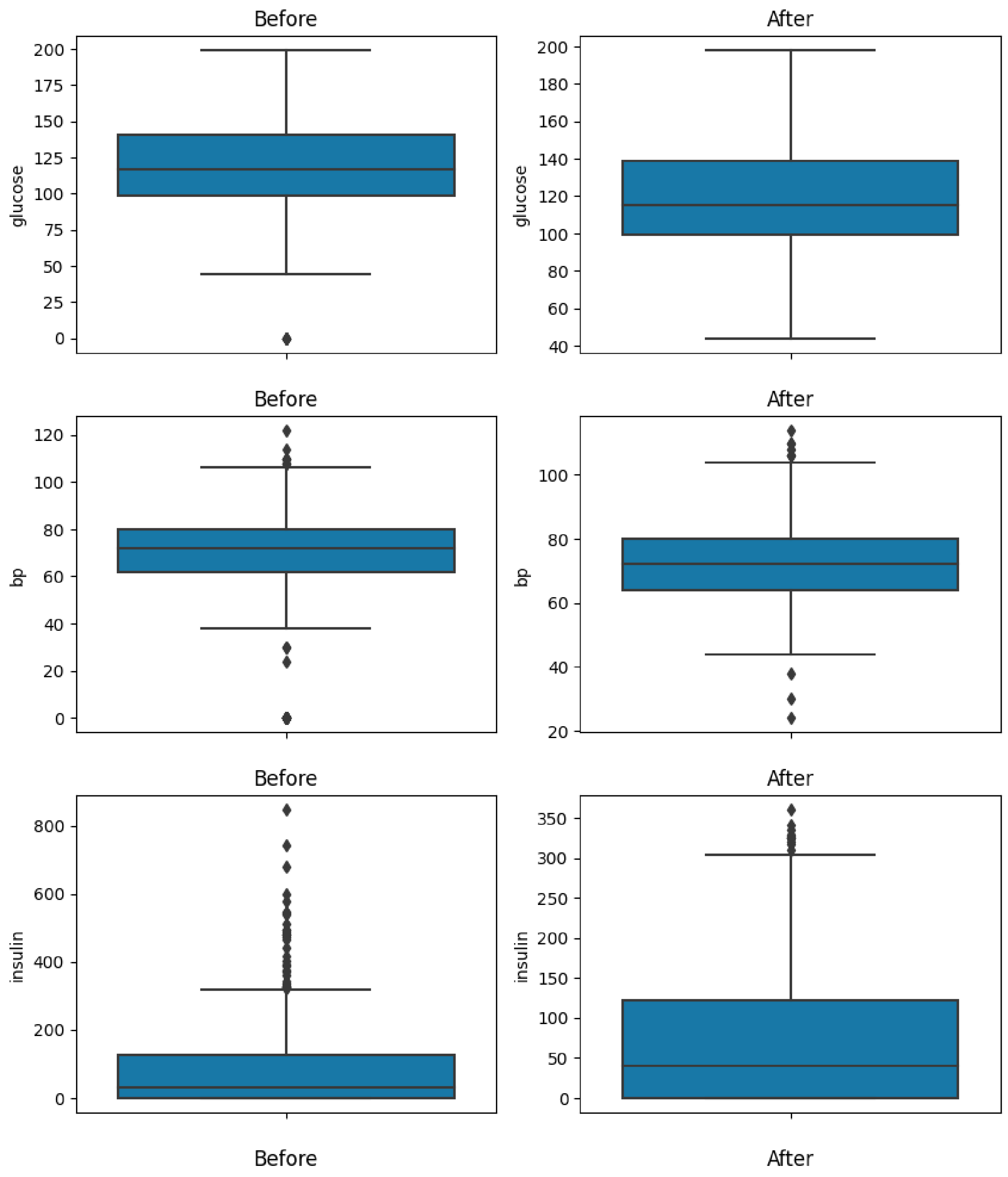
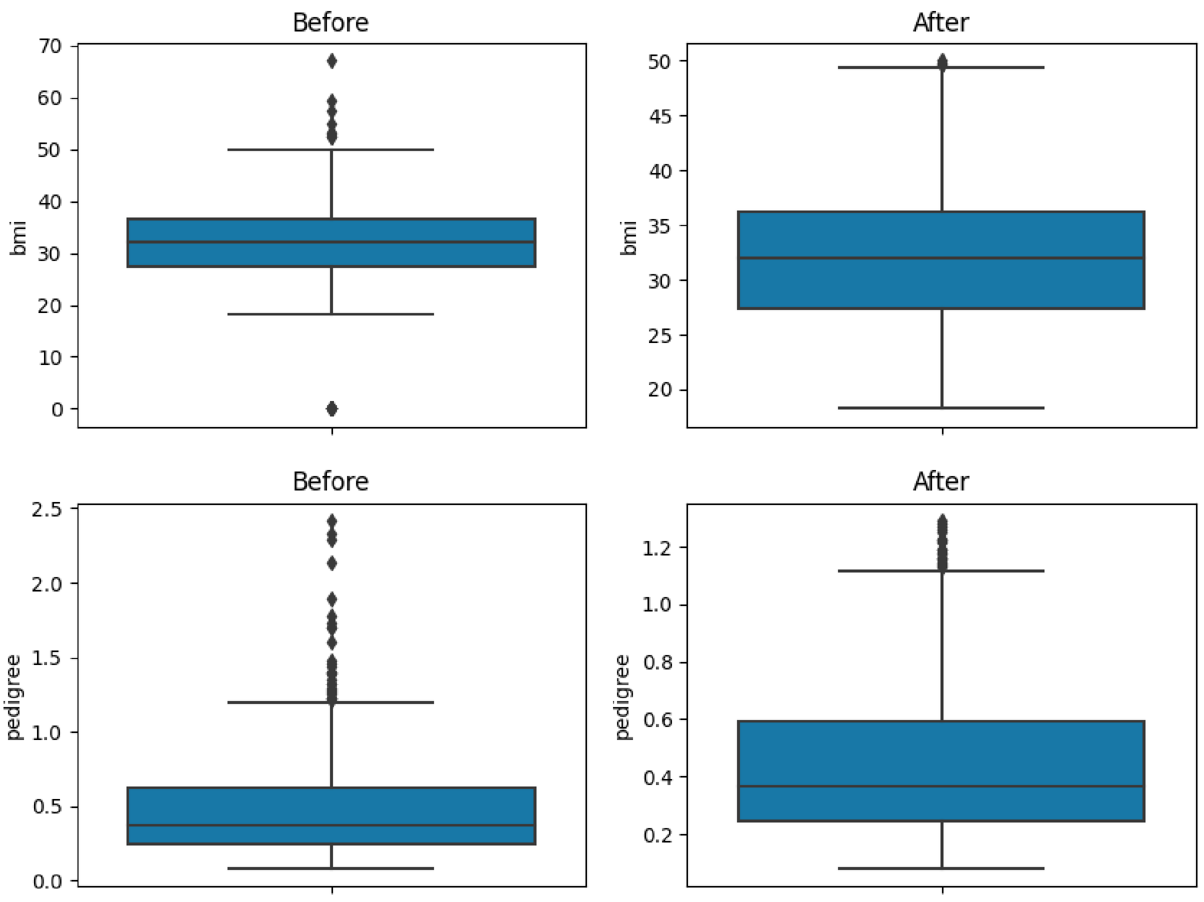


Fig 4: Comparison between distribution plot and boxplot of 5 attributes before and after handling outliers



boxplot of glucose, bp, insulin



boxplot of bmi, pedigree

Fig 5: Comparison boxplot of 5 attributes

From the distribution plots and box plots shown above, we can that a lot of outliers have been handled. Now we will use this data to train a Random Forest Classifier model. We will be training a Random Forest model to determine the principal attributes in the data set.

Model Building

Splitting data for training and testing purpose

We split the given train data set into two parts for training and testing purpose. The split ratio we used is 0.75 which indicates we used 75% data for training purpose and 25% data for testing purpose. We will be using the same split ratio for all models trained.

Random Forest Classifier Model
The object description of the Random Forest Classifier used is given below:

Object Name	Random Forest Classifier
Parameters	**Value**
Bootstrap	True
class_weight	None
Criterion	‘gini’
max_depth	None
max_features	‘auto’
max_leaf_nodes	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	2
oob_score	False
random_state	0
Verbose	1
warm_start	False

Table 6: Object Parameter Table for Random Forest Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Outcome / Actual Outcome	0	1
0	123	10
1	34	37

Table 7: Confusion matrix of Random Forest Classifier

We trained the Random Forest Classifier Model although it was not our goal, to find out the feature importance of the given attributes and choose the principal attributes for training our goal models. Random Forest Classifier has an inbuilt feature importance function that gives the value of importance of each attribute and helps in determining the target attribute.

The following table consists of the feature importance obtained from our Random Forest Classifier:

Attribute	**Feature Importance**
glucose	0.28611949728592045
Bp	0.10406010982541437
insulin	0.09215271103438498
bmi	0.18573718776000556
pedigree	0.1555065645601749
Age	0.17642392953409966

Table 8: Feature importance table obtained from Random Forest Classifier

As we can see from the calculated feature importance values, the attributes viz. bp and insulin have lower values compared to other attributes. So, we will remove these 2 attributes from our final data set, and remaining 4 will be our principal attributes to train our required models.

Attribute	**Feature Importance**
glucose	0.28611949728592045
bmi	0.18573718776000556
pedigree	0.1555065645601749
age	0.17642392953409966

Table 9: Feature importance of selected principal attributes

NOTE: These values will change with each run. But these 5 attributes will always have the highest feature importance if trained as above

Now we will modify the data set to include only the selected principal attributes. We made a copy of the initial data set and then dropped the non-principal attributes viz. ‘bp’ and ‘insulin’. Now we obtained a final data set which we will use for training purpose of our required models.

Given below is the description of the final data set obtained:

Please note, we have not included our target attribute ‘outcome’ in the data description.

	glucose	**Bmi**	**pedigree**	**age**	**outcome**
Type	Non-categorical	Non-categorical	Non-categorical	Non-categorical	Categorical
unique	Nan	Nan	Nan	Nan	Nan
Mean	120.070901	32.128965	0.441594	33.353028	0.327917
Standard Deviation	29.887179	6.457651	0.265363	11.771658	0.469802
Minimum	44.000000	18.200000	0.078000	21.000000	0.000000
25%	99.000000	27.400000	0.244000	24.000000	0.000000
50%	115.000000	32.009138	0.365000	29.000000	0.000000
75%	139.000000	36.200000	0.593000	41.000000	1.000000
Maximum	198.000000	50.000000	1.292000	81.000000	1.000000

Table 10: 5 Description of the final data obtained

Now we will be training our required models. Our project goal requires us to train specific 4 classifier models viz.

1. KNN Classifier
2. Naive Bayes Classifier
3. Decision Tree Classifier
4. Logistic Regression

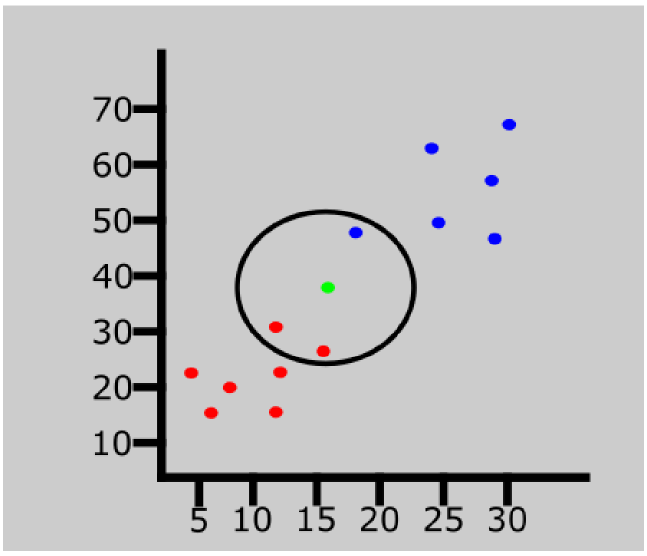
We will be using the final data set obtained after pre-processing the given train data set to train our required models.

KNN Classifier

k-NN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closet training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small).

For e.g. if k = 1, then the object is simply assigned to the class of that single nearest neighbor. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. The neighbors are taken from a set of objects for which the class (for kNN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.



An example of k-NN
classification:
Example of k-NN classification.
The test sample (green dot) should
be classified either to blue dot or
red dot. If k=3(solid line circle) it
is assigned to the red dot because
there are 2 red dots and only 1 blue
dot inside the circle.

We used a GridSearchCV object to find the best optimum value of k. Our test results gave the value of k = 7. The object description of the k-NN Classifier used is given below:

Object Name	KNeighborsClassifier
Parameters	**Value**
algorithm	‘auto’
leaf_size	30
metric	‘euclidean’
metric_params	None
n_jobs	-1
n_neighbors	5
p	2
weights	uniform

Table 11: Object Parameter Table for k-NN Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix.

Predicted Outcome / Actual Outcome	0	1
0	122	11
1	38	33

Table 12: Confusion matrix of k-NN Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained k-NN model. The ROC curve is given below:

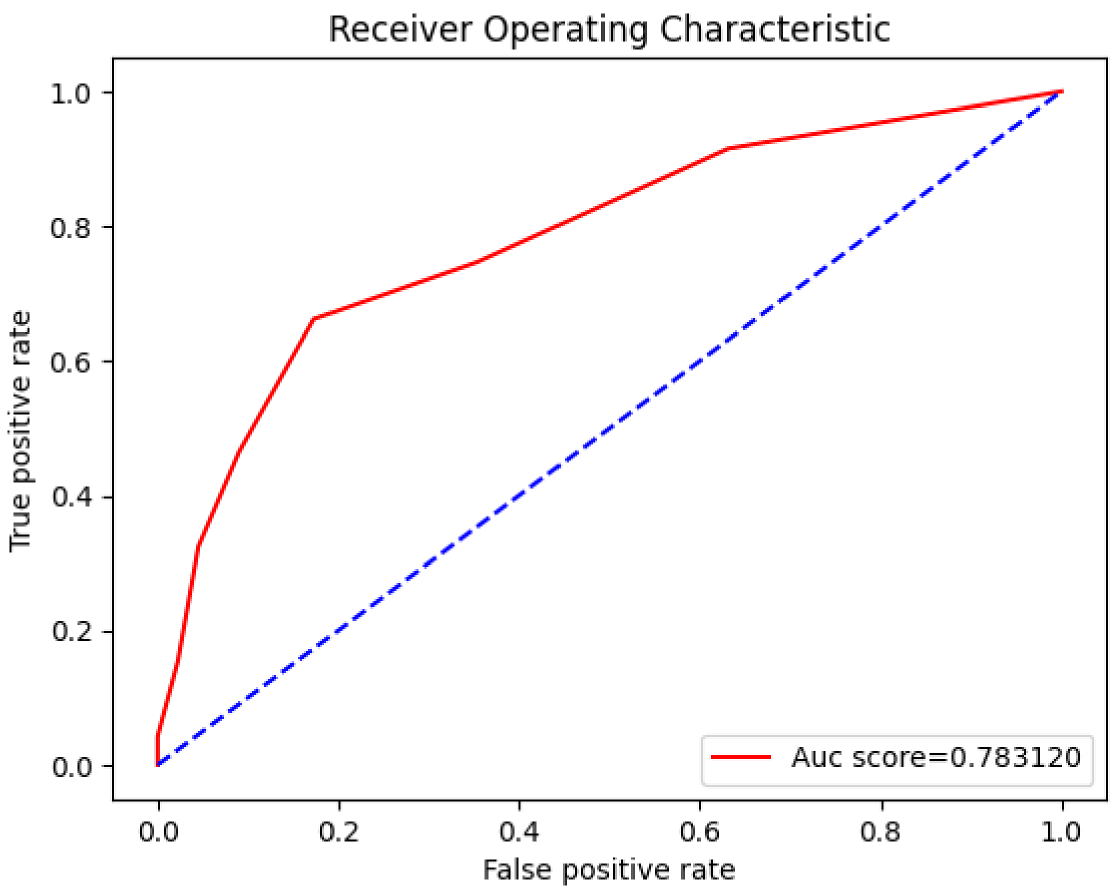


Fig 7: ROC curve of k-NN classifier model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our KNN classifier model is 0.783120

AUC value of KNN = 0.783120

Now we will be preparing the classification report of our k-NN model

Classification Report of k-NN classifier model				
Accuracy		0.76		
	precision	**recall**	**f1-score**	**support**
0	0.76	0.92	0.83	133
1	0.75	0.46	0.57	71

Table 13: Classification Report table of k-NN classifier

Naive Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem. The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

contribution to the outcome. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Bayes Theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = P(A).P(B|A)/P(B)$$

where A and B are events and P(B) is the probability of occurrence of event B. Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

P(A) is the priority of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B).

P(A|B) is a posterior probability of B, i.e. probability of event after evidence is seen.
The class-data relation from the Bayes Theorem can be obtained as follows:

$$P(Class|Data) = P(Class)\ P(Data|Class) / P(Data)$$

where,

P(Class|Data) = Posterior

P(Class) = Prior

P(Data|Class) = Likelihood

P(Data) = Marginal Probability

in other words, it can be written as:

$$Posterior = Prior * Likelihood / Marginal\ Probability$$

In application, we do not need to calculate the Marginal Probability for classification. We only need to calculate the numerator of the posterior for classification.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial Naive Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

The object description of the Naive Bayes used in given below:

Object Name	GaussianNB
Parameters	**Value**
Priors	None
var_smoothing	1e-09

Table 14: Object Parameter Table for Gaussian Naive Bayes Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Outcome	0	1
Actual Outcome		
0	125	8
1	34	37

Table 15: Confusion matrix of Gaussian Naive Bayes Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Gaussian Naive Bayes model. The ROC curve is given below:

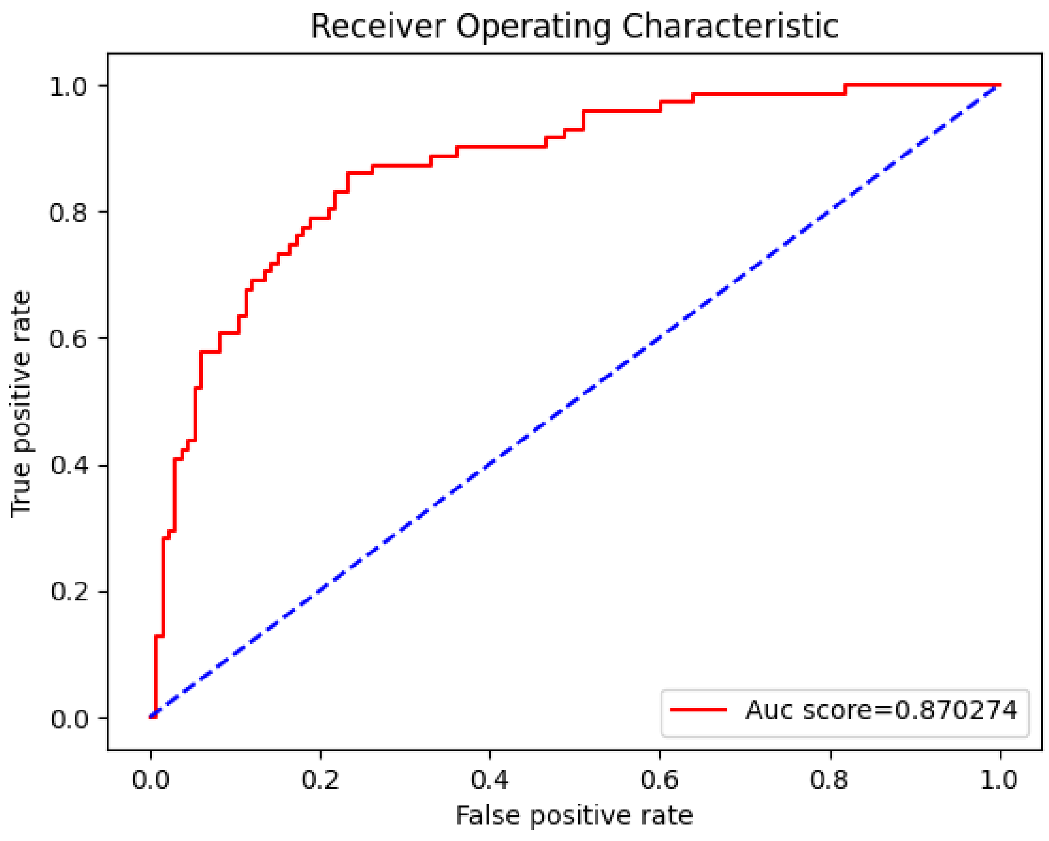


Fig 8: ROC curve of Gaussian Naive Bayes classifier model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Gaussian Naive Bayes model is 0.870274.

AUC value of Gaussian Naive Bayes = 0.870274.
Now we will be preparing the classification report of our Gaussian Naive Bayes model

Classification Report of Gaussian Naive Bayes model				
Accuracy			0.75	
	precision	**recall**	**f1-score**	**support**
0	0.79	0.94	0.86	133
1	0.82	0.52	0.64	71

Table 16: Classification Report table of Gaussian Naive classifier

Decision Tree

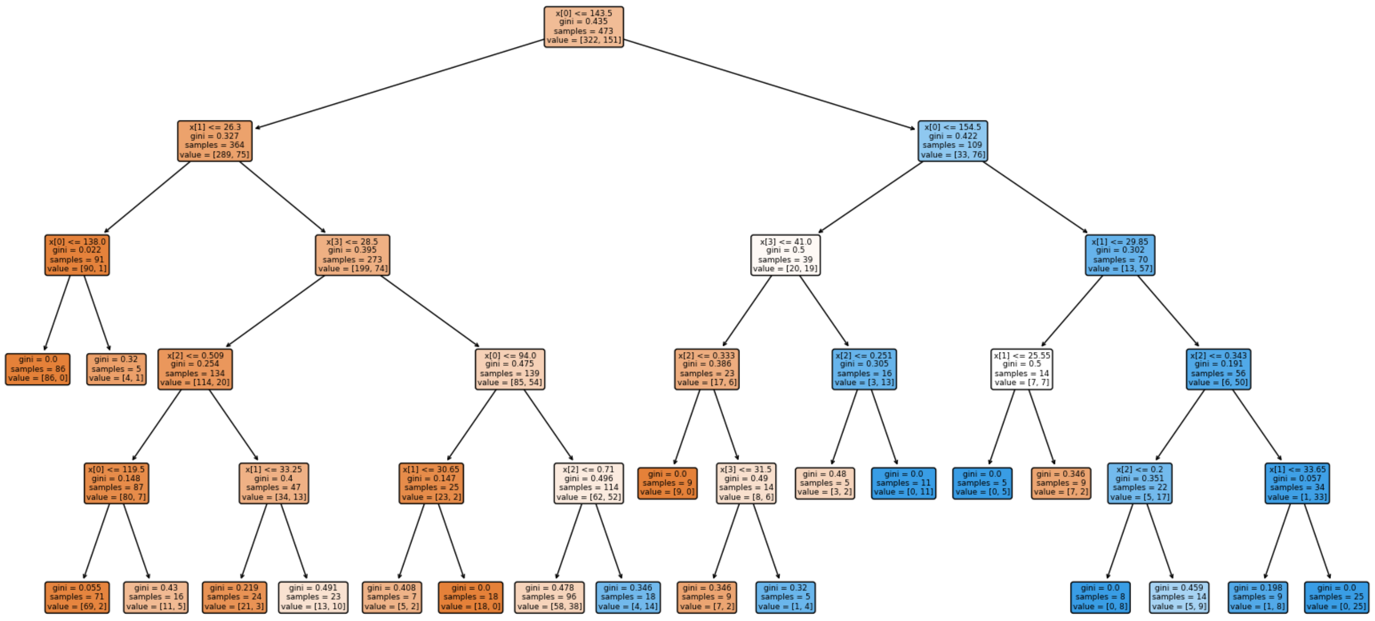
Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, it's also widely used in machine learning.

The object description of the Decision Tree Classifier used is given below:

Object Name	Decision TreeClassifier
Parameters	**Value**
ccp_alpha	0.0
class_weight	None
Criterion	Gini
max_depth	5
max_features	None
max_leaf_nodes	None
min_impurity_decrease	0.0
min_samples_leaf	5
min_samples_split	2
min_weight_fraction_leaf	0.0
random_state	100
Splitter	Best

Table 17: Object parameter Table for Decision Tree Classifier

Image of Decision Tree:


x[0] <= 143.5 gini = 0.435 samples = 473 value = [322, 151]
x[1] <= 26.3 gini = 0.327 samples = 364 value = [289, 75]
x[0] <= 154.5 gini = 0.422 samples = 109 value = [33, 76]
x[0] <= 138.0 gini = 0.022 samples = 91 value = [90, 1]
x[3] <= 28.5 gini = 0.395 samples = 273 value = [199, 74]
x[3] <= 41.0 gini = 0.5 samples = 39 value = [20, 19]
x[1] <= 29.85 gini = 0.302 samples = 70 value = [13, 57]
gini = 0.0 samples = 86 value = [86, 0]
gini = 0.32 samples = 5 value = [4, 1]
x[2] <= 0.509 gini = 0.254 samples = 134 value = [114, 20]
x[0] <= 94.0 gini = 0.475 samples = 139 value = [85, 54]
x[2] <= 0.333 gini = 0.386 samples = 23 value = [17, 6]
x[2] <= 0.251 gini = 0.305 samples = 16 value = [3, 13]
x[1] <= 25.55 gini = 0.5 samples = 14 value = [7, 7]
x[2] <= 0.343 gini = 0.191 samples = 56 value = [6, 50]
x[0] <= 119.5 gini = 0.148 samples = 87 value = [80, 7]
x[1] <= 33.25 gini = 0.4 samples = 47 value = [34, 13]
x[1] <= 30.65 gini = 0.147 samples = 25 value = [23, 2]
x[2] <= 0.71 gini = 0.496 samples = 114 value = [62, 52]
gini = 0.0 samples = 9 value = [9, 0]
x[3] <= 31.5 gini = 0.49 samples = 14 value = [8, 6]
gini = 0.48 samples = 5 value = [3, 2]
gini = 0.0 samples = 11 value = [0, 11]
gini = 0.0 samples = 5 value = [0, 5]
gini = 0.346 samples = 9 value = [7, 2]
x[2] <= 0.2 gini = 0.351 samples = 22 value = [5, 17]
x[1] <= 33.65 gini = 0.057 samples = 34 value = [1, 33]
gini = 0.055 samples = 71 value = [69, 2]
gini = 0.43 samples = 16 value = [11, 5]
gini = 0.219 samples = 24 value = [21, 3]
gini = 0.491 samples = 23 value = [13, 10]
gini = 0.408 samples = 7 value = [5, 2]
gini = 0.0 samples = 18 value = [18, 0]
gini = 0.478 samples = 96 value = [58, 38]
gini = 0.346 samples = 18 value = [4, 14]
gini = 0.346 samples = 9 value = [7, 2]
gini = 0.32 samples = 5 value = [1, 4]
gini = 0.0 samples = 8 value = [0, 8]
gini = 0.459 samples = 14 value = [5, 9]
gini = 0.198 samples = 9 value = [1, 8]
gini = 0.0 samples = 25 value = [0, 25]

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Outcome / Actual Outcome	0	1
0	123	10
1	40	31

Table 18: Confusion matrix of Decision Tree Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Decision Tree model. The ROC curve is given below:

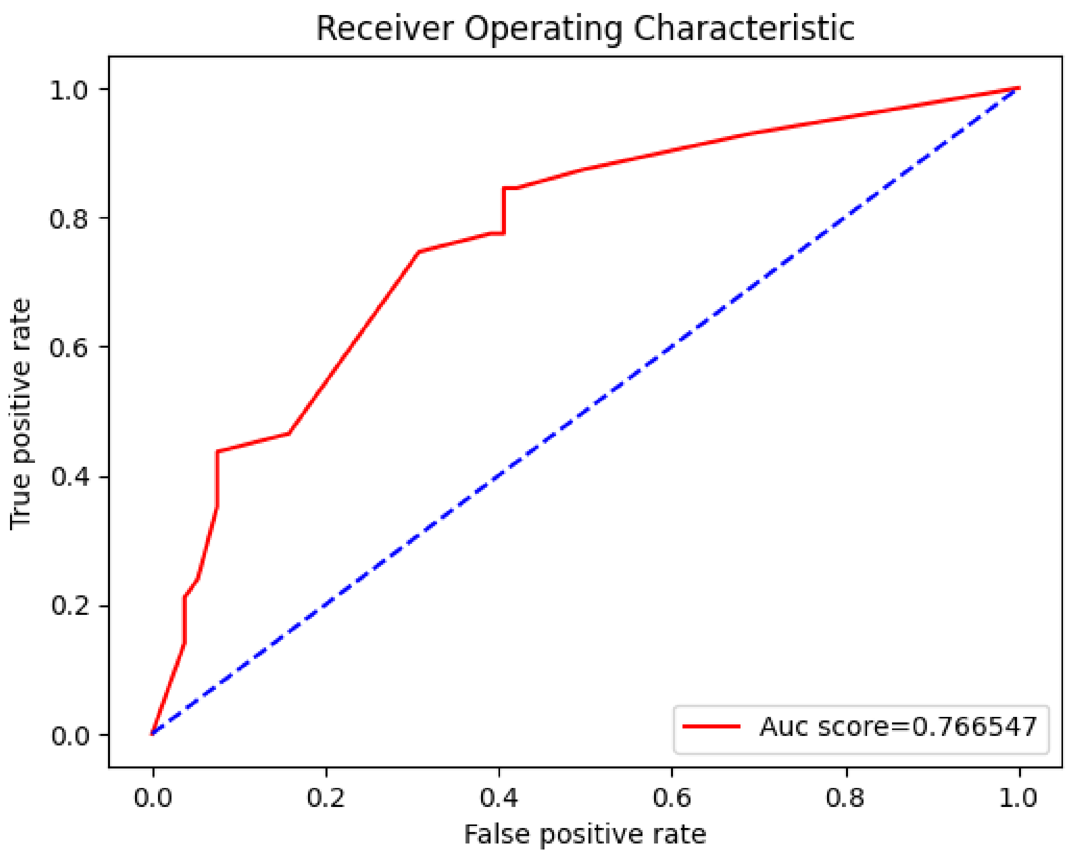


Fig 9: ROC curve of Decision Tree model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Decision Tree classifier model is 0.766547.

AUC value of Decision Tree = 0.766547.

Now we will be preparing the classification report of our Decision Tree

Classification Report of Decision Tree Classifier model				
Accuracy			0.75	
	precision	**recall**	**f1-score**	**support**
0	0.75	0.92	0.83	133
1	0.76	0.44	0.55	71

Table 19: Classification Report table of Decision Tree classifier

Logistic Regression

Logistic Regression (also called Logistic Regression) is commonly used to estimate the probability that an instance belongs to a particular class (e.g., what is the probability that this email is spam?). If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled "1"), or else it predicts that it does not (i.e., it belongs to the negative class, labeled "0"). This makes it a binary classifier.

Binary logistic regression major assumptions:

- The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
- There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
- There should be no high correlations (multi co linearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Some authors suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.

At the center of the logistic regression analysis is the task estimating the log odds of an event. Mathematically, logistic regression estimates a multiple linear regression function defined as: logit(p)

$$= \log(\ p(y=1)\ /\ 1-(p=1)\ ) = ß_0 + ß_1.x_a + ß_2.x_a + \ldots + ß_n.x_a$$

for i =1.....n.

The object description of the Logistic Regression Classifier used given below:

Object Name	**LogisticRegression**
Parameters	**Value**
C	1.0
class_weight	None
Dual	False
fit_intercept	True
intercept_scaling	1
l1_ratio	None
max_iter	100
multi_class	Auto
n_jobs	None
penalty	12
random_state	None
solver	Lbfgs
tol	0.0001
verbose	0
warm_start	False

Table 20: Object Parameter Table for Logistic Regression Classifier

The intercept of the trained model is: -9.270798

The coefficients of the model are given in the table below:

Attributes	**Coefficient**
glucose	0.0367178
bmi	0.07746383
pedigree	1.17643188
age	0.02473828

Table 21: Coefficients of attributes if the trained Logistic Regression Model

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Outcome	0	1
Actual Outcome		
0	123	10
1	40	31

Table 18: Confusion matrix of Decision Tree Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Logistic Regression model. The ROC curve is given below:

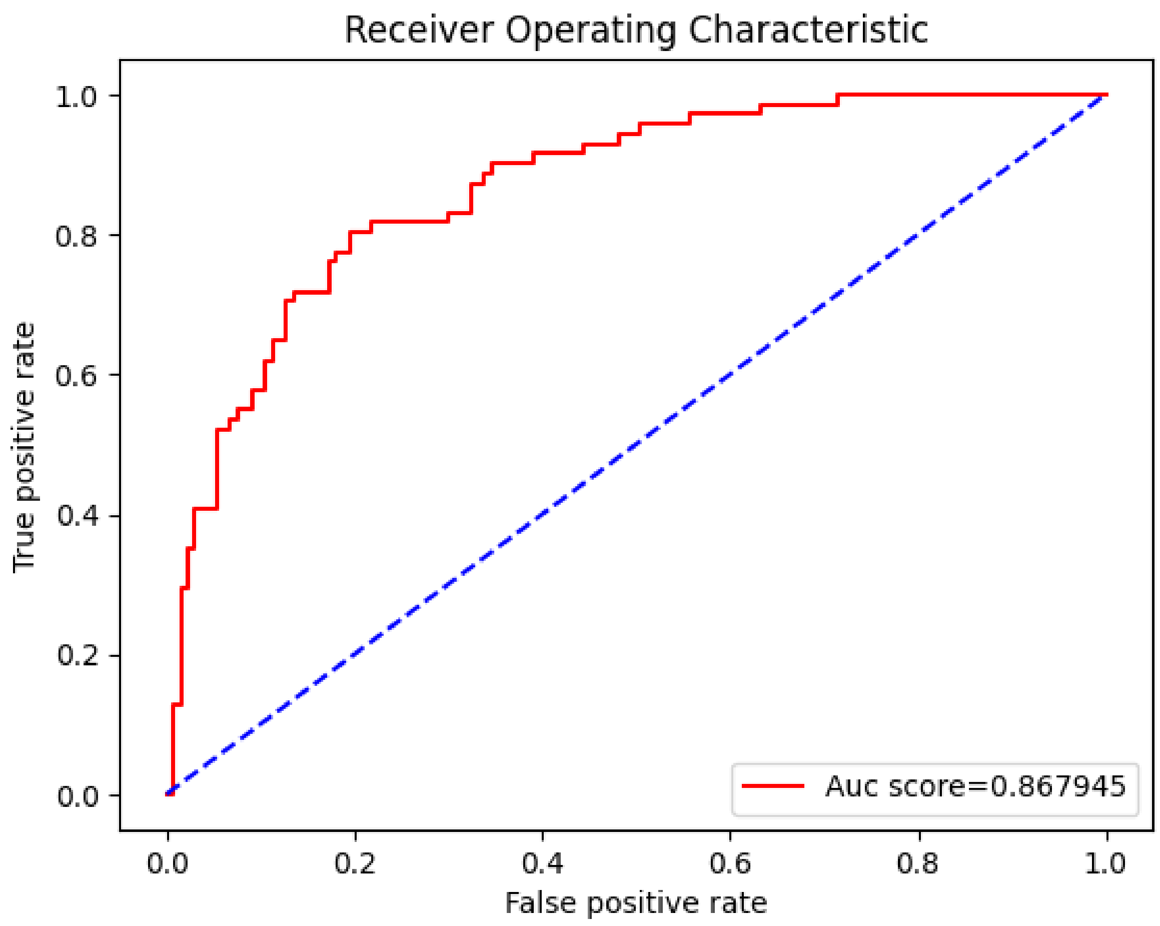


Fig 11: ROC curve of Logistic Regression model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Logistic Regression model is 0.867945.

AUC value of Logistic Regression = 0.867945.

Now we will be preparing the classification report of our Logistic Regression model.

Classification Report of Logistic Regression Classifier model				
Accuracy			0.78	
	precision	**recall**	**f1-score**	**support**
0	0.76	0.95	0.85	133
1	0.82	0.46	0.59	71
Logistic Regression RMSE			0.46966821831386213	
Logistic Regression R-squared			0.027851318436937555	

Table 23: Classification Report table of Decision Tree classifier

Comparison of the Models trained

We trained 4 models using the 4 algorithms viz.

1. k-Nearest Neighbour
2. Gaussian Naive Bayes
3. Decision Tree and
4. Logistic Regression

The 4 models had different accuracy. The comparison of the accuracies of the models are given below:

Model	**Accuracy**
k-Nearest Neighbour	0.76
Gaussian Naive Bayes	0.79
Decision Tree	0.75
Logistic Regression	0.78

Table 24: Accuracy Comparison Table

The following bar graph shows the accuracy comparison in graphical way:

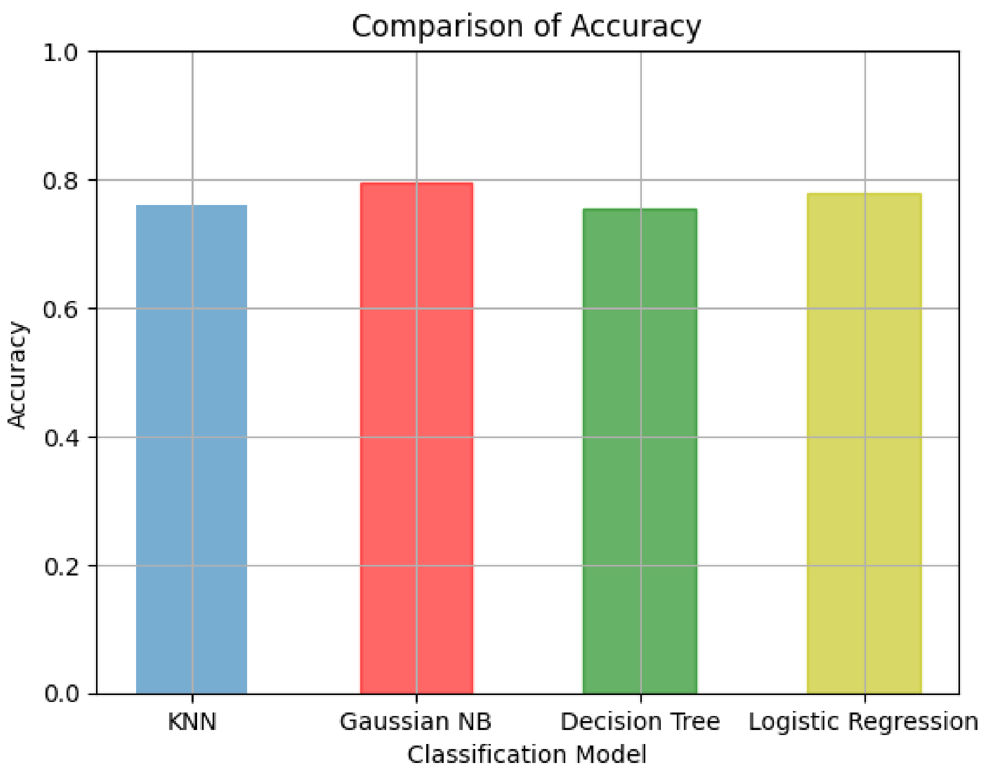


Fig 12: Comparison of accuracy of the 4 different models trained

Thus, from the above comparison we can see that Gaussian Naive Bayes classifier gives the highest accuracy while training and testing with our data set.

We choose the following classifier models as they were most accurate and in accordance with each other:

Classifier Model Trained	**Data Used**	**Accuracy%**
K-NN	Removing outliers, without standardizing	0.75
Gaussian Naive Bayes	Removing Outliers	0.79
Logistic Regression	Removing Outliers	0.78
Decision Tree	Removing Outliers	0.75

Table 25: Selected classifier models and data set use

We also trained the above-mentioned classifier without removing outliers from the data set. The following table shows the comparison of accuracy of the classifiers trained:

Classifier Model Trained	**Data Used**	**Accuracy**
k-NN	Removing outliers, without standardizing	0.75
k-NN	Removing outliers, with standardizing	0.75
k-NN	With outliers, without standardizing	0.71
k-NN	With outliers, with standardizing	0.72
Gaussian Naive Bayes	With outliers	0.74
Gaussian Naive Bayes	Removing outliers	0.79
Logistic Regression	With outliers	0.74
Logistic Regression	Removing outliers	0.78
Decision Tree	With outliers	0.66
Decision Tree	Removing outliers	0.75

Table 26: Comparison of different models trained with different version of the data set

diabetes-prediction-project

1 Diabetes Prediction System

Importing Modules

```
[4]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Loading dataset into **Pandas dataframe**

```
[5]: #Loading data into pandas dataframe
Data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv")

#making a copy of main dataset
X = Data.copy()
X.shape
```

```
[5]: (768, 7)
```

```
[6]: X.head(10)
```

```
[6]:   glucose    bp  insulin   bmi  pedigree age  outcome
    0      148  72.0        0  33.6     0.627  50     Yes
    1       85  66.0        0  26.6     0.351  31      No
    2      183  64.0        0  23.3     0.672  32     Yes
    3       89  66.0       94  28.1     0.167  21      No
    4      137  40.0      168  43.1     2.288  33     Yes
    5      116  74.0        0  25.6     0.201  30      No
    6       78  50.0       88  31.0     0.248  26     Yes
    7      115   0.0        0  35.3     0.134  29      No
    8      197  70.0      543  30.5     0.158  53     Yes
    9      125  96.0        0   0.0     0.232  54     Yes
```

From the data description, we can see that there are null values in columns bp and bmi. Also there are non-numeric columns. So we need to pre-process the data before using it to train any model.

Pre-Processing Data

Changing non-numeric values to numeric values

```
[7]: X.outcome.replace(['No','Yes'],[0,1], inplace=True) #replacing 'No'
      with 0 and␣ ↪'Yes' with 1 in 'Outcome'
     X.head(10)
```

```
[7]:   glucose    bp  insulin   bmi  pedigree  age outcome
     0     148  72.0        0  33.6     0.627   50        1
     1      85  66.0        0  26.6     0.351   31        0
     2     183  64.0        0  23.3     0.672   32        1
     3      89  66.0       94  28.1     0.167   21        0
     4     137  40.0      168  43.1     2.288   33        1
     5     116  74.0        0  25.6     0.201   30        0
     6      78  50.0       88  31.0     0.248   26        1
     7     115   0.0        0  35.3     0.134   29        0
     8     197  70.0      543  30.5     0.158   53        1
     9     125  96.0        0   0.0     0.232   54        1
```

Searching for Null Values

```
[8]: print("Null values count:\n",X.isnull().sum()) #sum of Null values in a column
```

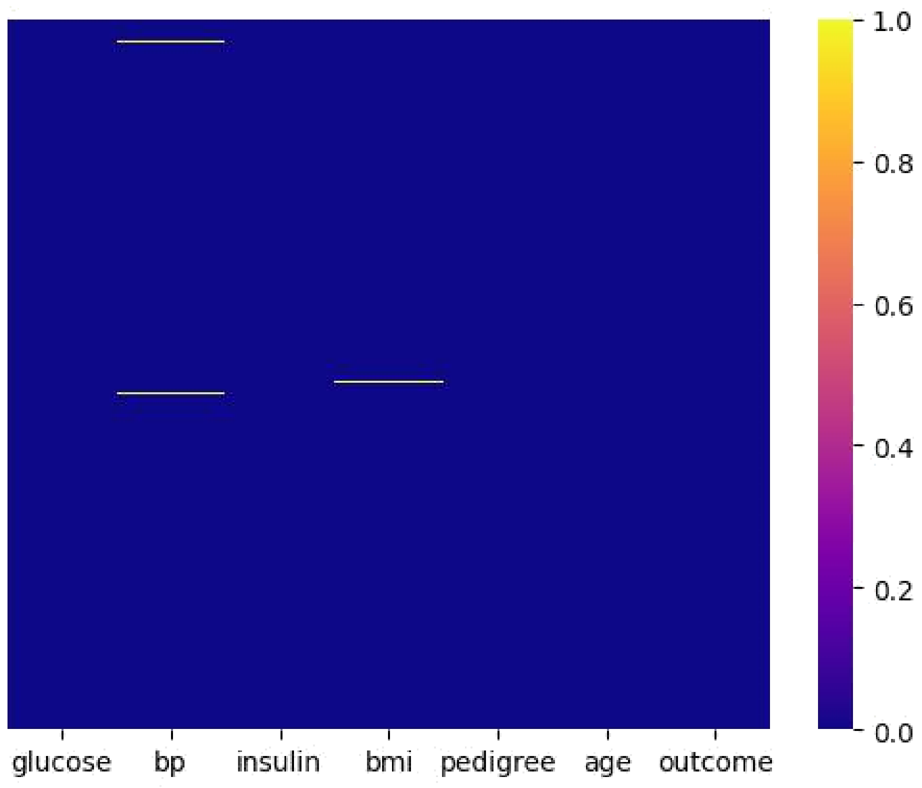
```
Null values count:
 glucose    0
bp         2
insulin    0
bmi        2
pedigree   0
age        0
outcome    0
dtype: int64
```

We can see that dataset had null values. Let us visualise the null values using

heatmap. **Visualising Null values using heatmap**

```
[9]: sns.heatmap(X.isnull(),yticklabels=False,cbar=True,cmap='plasma')
```

```
[9]: <Axes: >
```

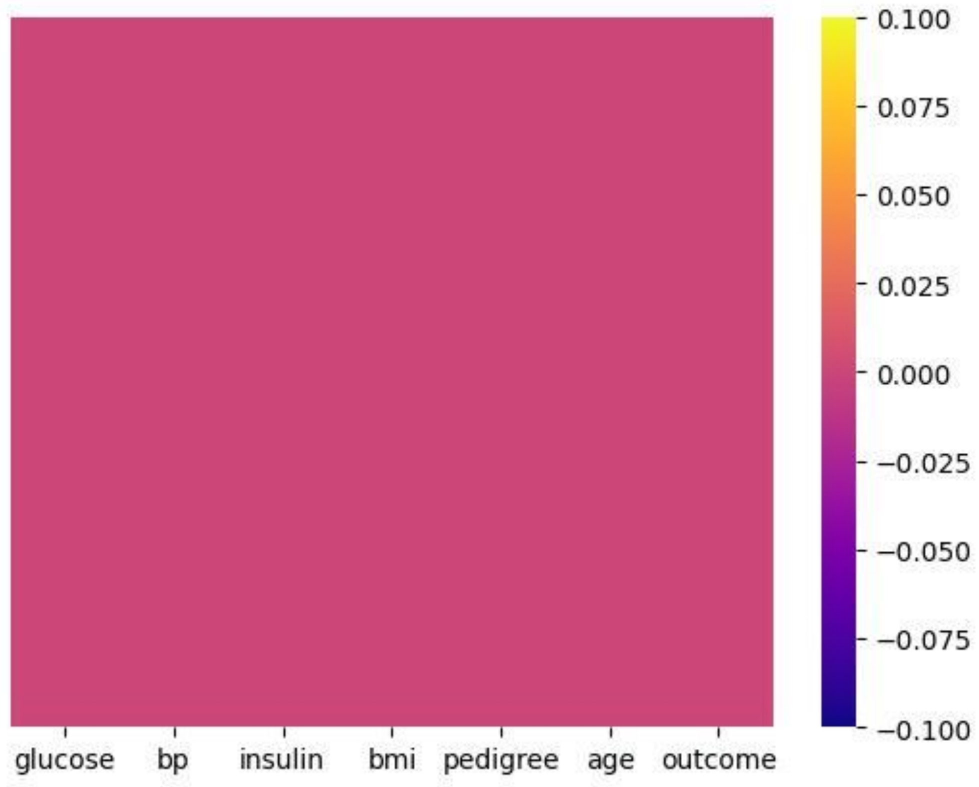


Handling null values

```
[10]:  X['bmi'].fillna(X['bmi'].mean(), inplace = True)
X['bp'].fillna(X['bp'].mean(), inplace = True)

sns.heatmap(X.isnull(),yticklabels=False,cbar=True,cmap='plasma')
X.describe()
```

```
[10]:          glucose          bp     insulin         bmi    pedigree         age  \
count   768.000000  768.000000  768.000000  768.000000  768.000000  768.000000
mean    120.894531   69.079634   79.799479   32.009138    0.471876   33.240885
std      31.972618   19.334027  115.244002    7.876858    0.331329   11.760232
min       0.000000    0.000000    0.000000    0.000000    0.078000   21.000000
25%      99.000000   62.000000    0.000000   27.300000    0.243750   24.000000
50%     117.000000   72.000000   30.500000   32.004569    0.372500   29.000000
75%     140.250000   80.000000  127.250000   36.600000    0.626250   41.000000
max     199.000000  122.000000  846.000000   67.100000    2.420000   81.000000

          outcome
count  768.000000
mean     0.348958
std      0.476951
```

```
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      1.000000
```



Now we have successfully handled Null values and converted non-numeric values to Numeric valus. So we're moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'Outcome' column in the dataset

Checking Correlation of every attribute with our target attribute i.e 'Outcome'

```
[11]:  correlation = X.corr()
print("Correlation of each attribute with Outcome: \n\n",correlation['outcome'])
```

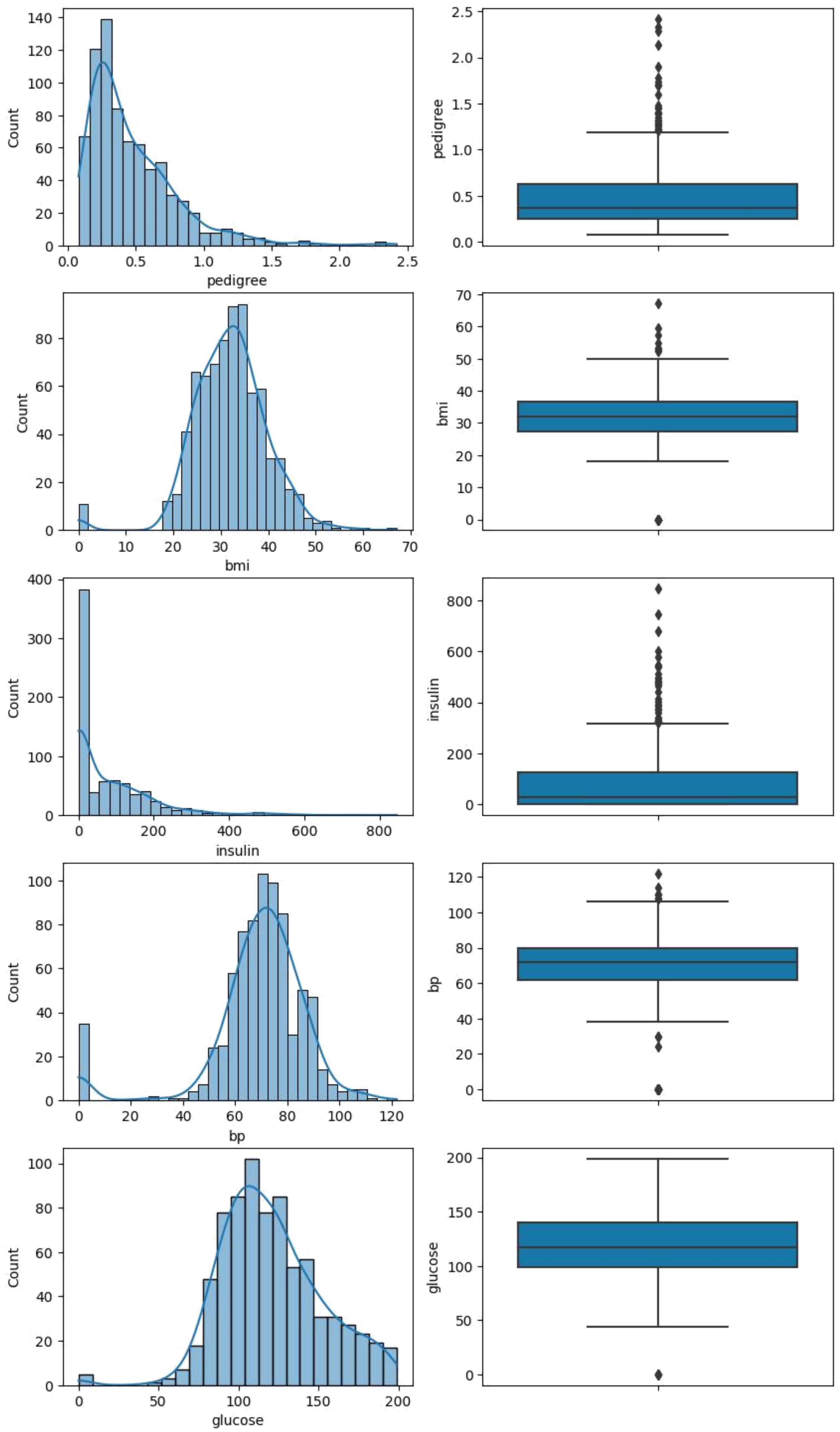
```
Correlation of each attribute with Outcome:

 glucose    0.466581
bp         0.063315
insulin    0.130548
bmi        0.292956
pedigree   0.173844
age        0.238356
```

```
outcome    1.000000
Name: outcome, dtype: float64
```

Checking for outliers

```
[12]:  #plotting distribution plot and boxplot for

f,axes = plt.subplots(5,2,figsize=(10,18))

sns.histplot(ax=axes[0][0],data=X,x='pedigree',kde=True)
sns.boxplot(ax=axes[0][1],data=X,y="pedigree",palette="winter")

sns.histplot(ax=axes[1][0],data=X,x='bmi',kde=True)
sns.boxplot(ax=axes[1][1],data=X,y="bmi",palette="winter")

sns.histplot(ax=axes[2][0],data=X,x='insulin',kde=True)
sns.boxplot(ax=axes[2][1],data=X,y="insulin",palette="winter")

sns.histplot(ax=axes[3][0],data=X,x='bp',kde=True)
sns.boxplot(ax=axes[3][1],data=X,y="bp",palette="winter")

sns.histplot(ax=axes[4][0],data=X,x='glucose',kde=True)
sns.boxplot(ax=axes[4][1],data=X,y="glucose",palette="winter")
```

```
[12]: <Axes: ylabel='glucose'>
```


Count
pedigree
Count
bmi
Count
insulin
Count
bp
Count
glucose

From the distribution plots and boxplots we can see that there are possible outliers in the columns ‘glucose’, ‘bp’, ‘insulin’, ‘bmi’ and ‘pedigree’.

We have not checked for outliers in the column outcome as it contains values in binaries and

Handling the outliers

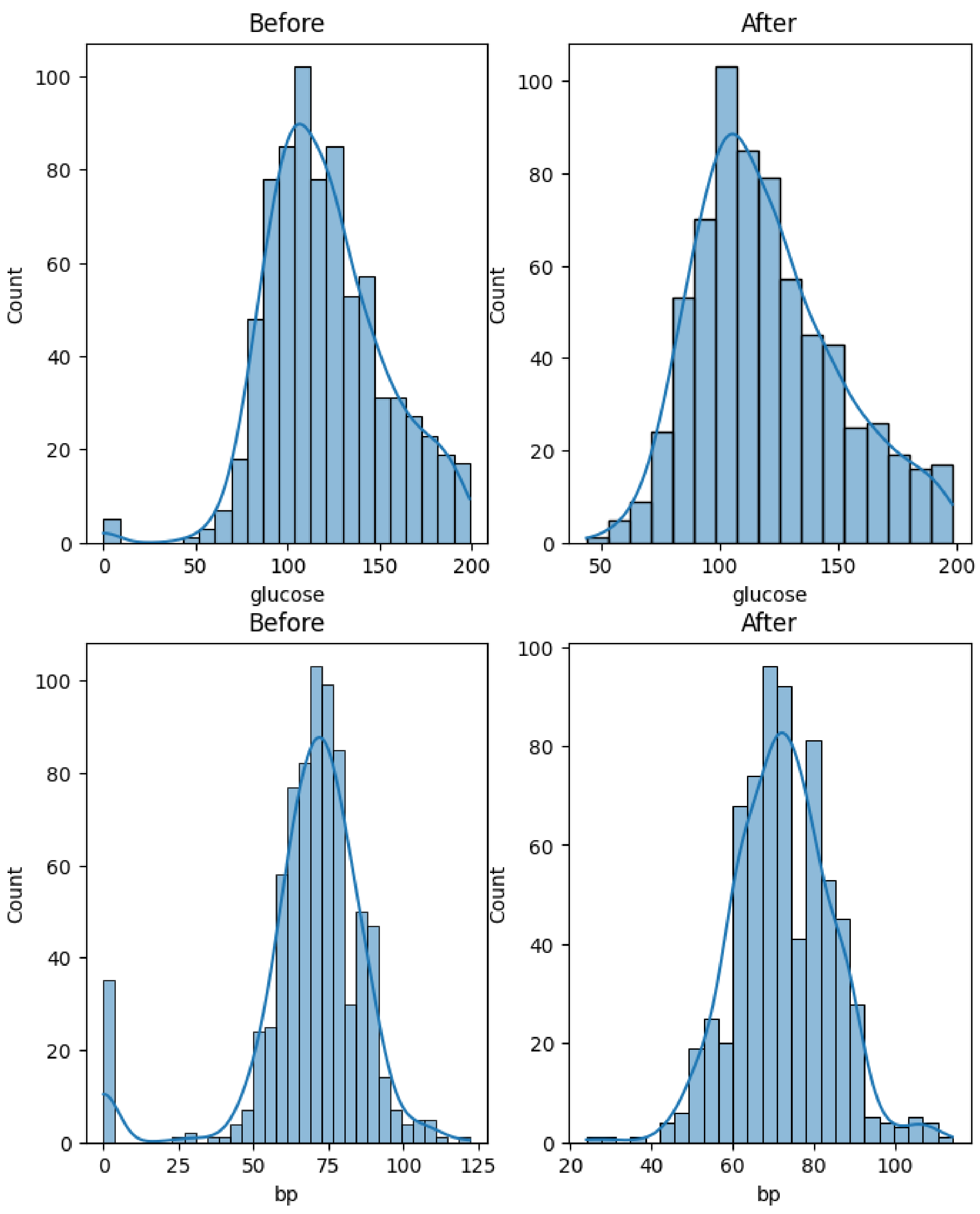
We used Z-score method to handle the outliers

```
[13]:  from scipy import stats

X_having_outliers = X.copy() #making copy of dataset having outliers


#To calculating Z-score we've to subtract any random data-point with

  it's mean␣ ↪then have to divide with standard deviation
#getting the z-score of glucose attribute as an
numpy array glu_z = np.abs(stats.zscore(X.glucose))

a1 = np.array(np.where(glu_z > 2.5)).flatten() #checking indexes

  having z-score␣ ↪greater than 2.5 and making an array of it
#getting the z-score of bp attribute as an
numpy array bp_z = np.abs(stats.zscore(X.bp))

a2 = np.array(np.where(bp_z > 2.5)).flatten() #checking indexes

  having z-score␣ ↪greater than 2.5 and making an array of it
#getting the z-score of insulin attribute as an
numpy array ins_z = np.abs(stats.zscore(X.insulin))

a3 = np.array(np.where(ins_z > 2.5)).flatten() #checking indexes

  having z-score␣ ↪greater than 2.5 and making an array of it
#getting the z-score of bmi attribute as an
numpy array bmi_z = np.abs(stats.zscore(X.bmi))

a4 = np.array(np.where(bmi_z > 2.5)).flatten() #checking indexes

  having z-score␣ ↪greater than 2.5 and making an array of it
#getting the z-score of pedigree attribute as an numpy
array pedigree_z = np.abs(stats.zscore(X.pedigree))

a5 = np.array(np.where(pedigree_z > 2.5)).flatten() #checking

  indexes having␣ ↪z-score greater than 2.5 and making an array of it
#merging all 1d array into one
b1 = np.concatenate((a1,a2,a3,a4,a5))

#dropping indexes that has the z-score greater
than 2.5 X.drop(b1,axis=0,inplace=True)
```

Comparison of dataset before and after removing the outliers

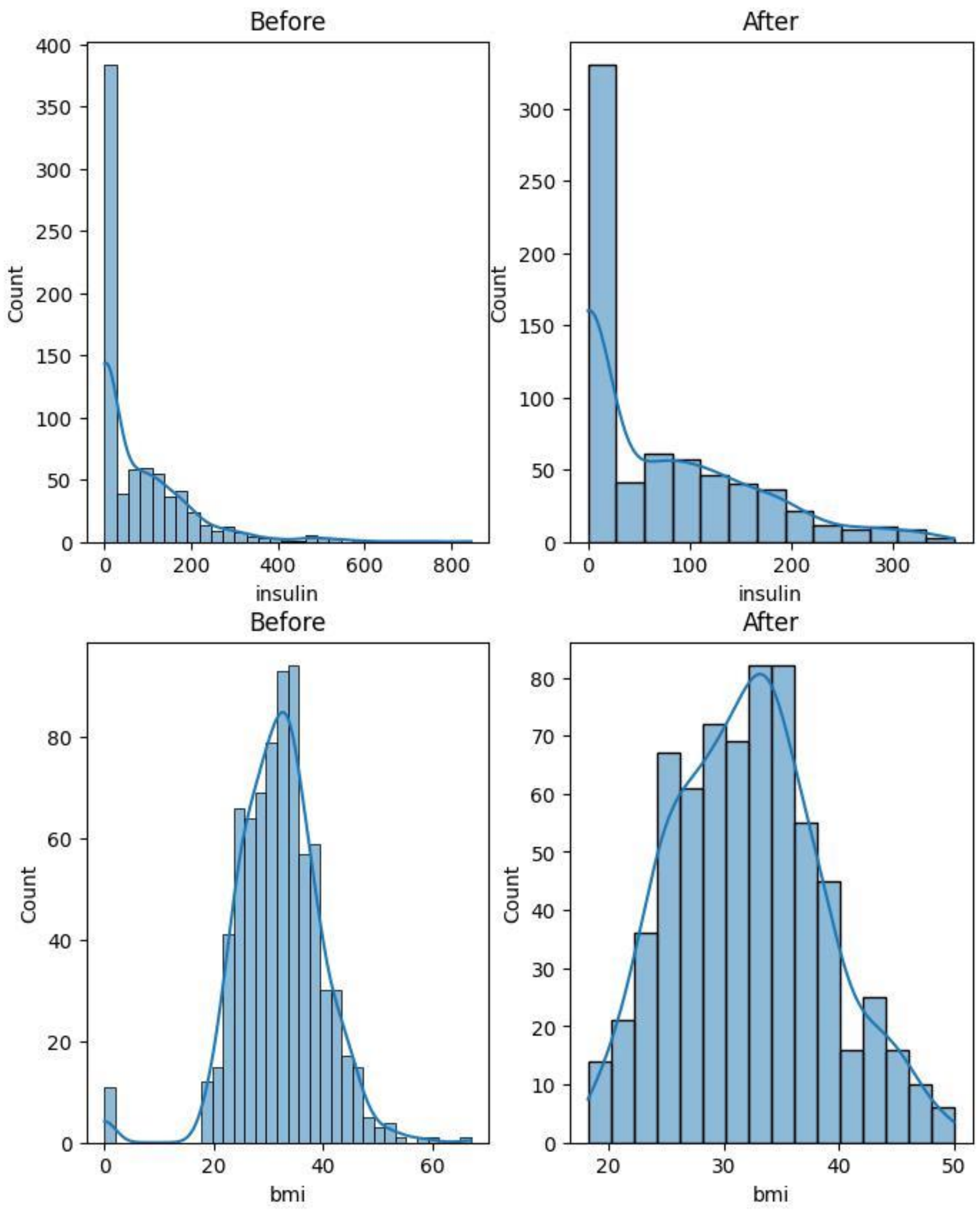
```
[14]: #comparing distributions of with outliers and without
      outliers f,axes = plt.subplots(2,2,figsize=(8,10))
      sns.histplot(ax=axes[0][0],data=X_having_outliers,x="glucose",kde=True).
       ↪set_title("Before")
      sns.histplot(ax=axes[0][1],data=X,x="glucose",kde=True).set_title("After")

      sns.histplot(ax=axes[1][0],data=X_having_outliers,x="bp",kde=True).
       ↪set_title("Before")
      sns.histplot(ax=axes[1][1],data=X,x="bp",kde=True).set_title("After")
```

```
[14]: Text(0.5, 1.0, 'After')
```



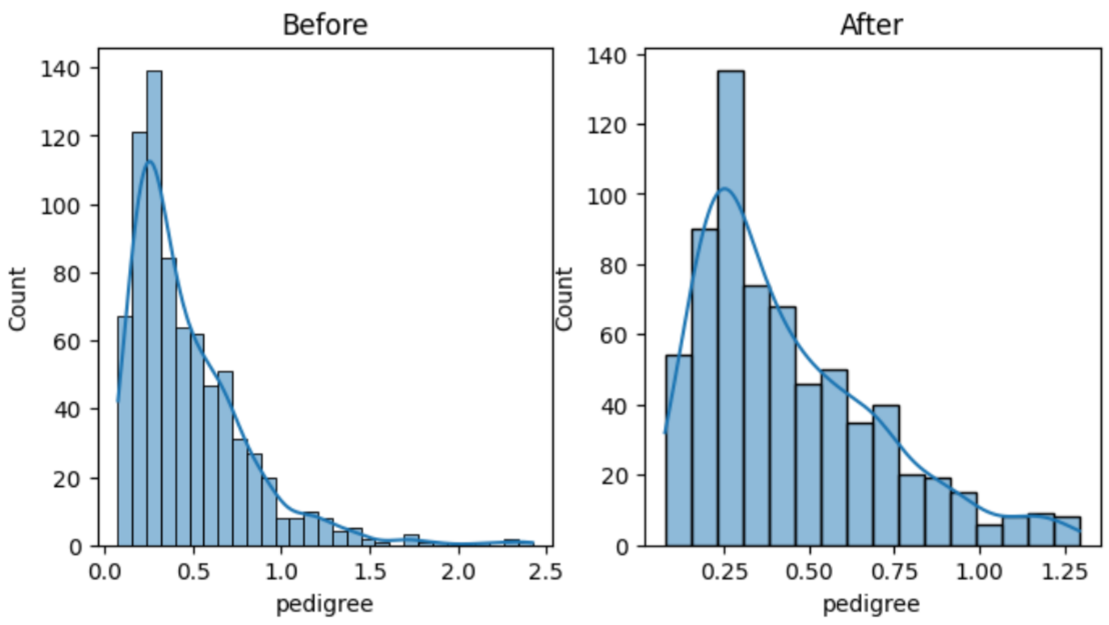
```
[15]:  f,axes = plt.subplots(2,2,figsize=(8,10))
sns.histplot(ax=axes[0][0],data=X_having_outliers,x="insulin",kde=True).
 ↪set_title("Before")
sns.histplot(ax=axes[0][1],data=X,x="insulin",kde=True).set_title("After")
sns.histplot(ax=axes[1][0],data=X_having_outliers,x="bmi",kde=True).
 ↪set_title("Before")
sns.histplot(ax=axes[1][1],data=X,x="bmi",kde=True).set_title("After")
```

```
[15]: Text(0.5, 1.0, 'After')
```



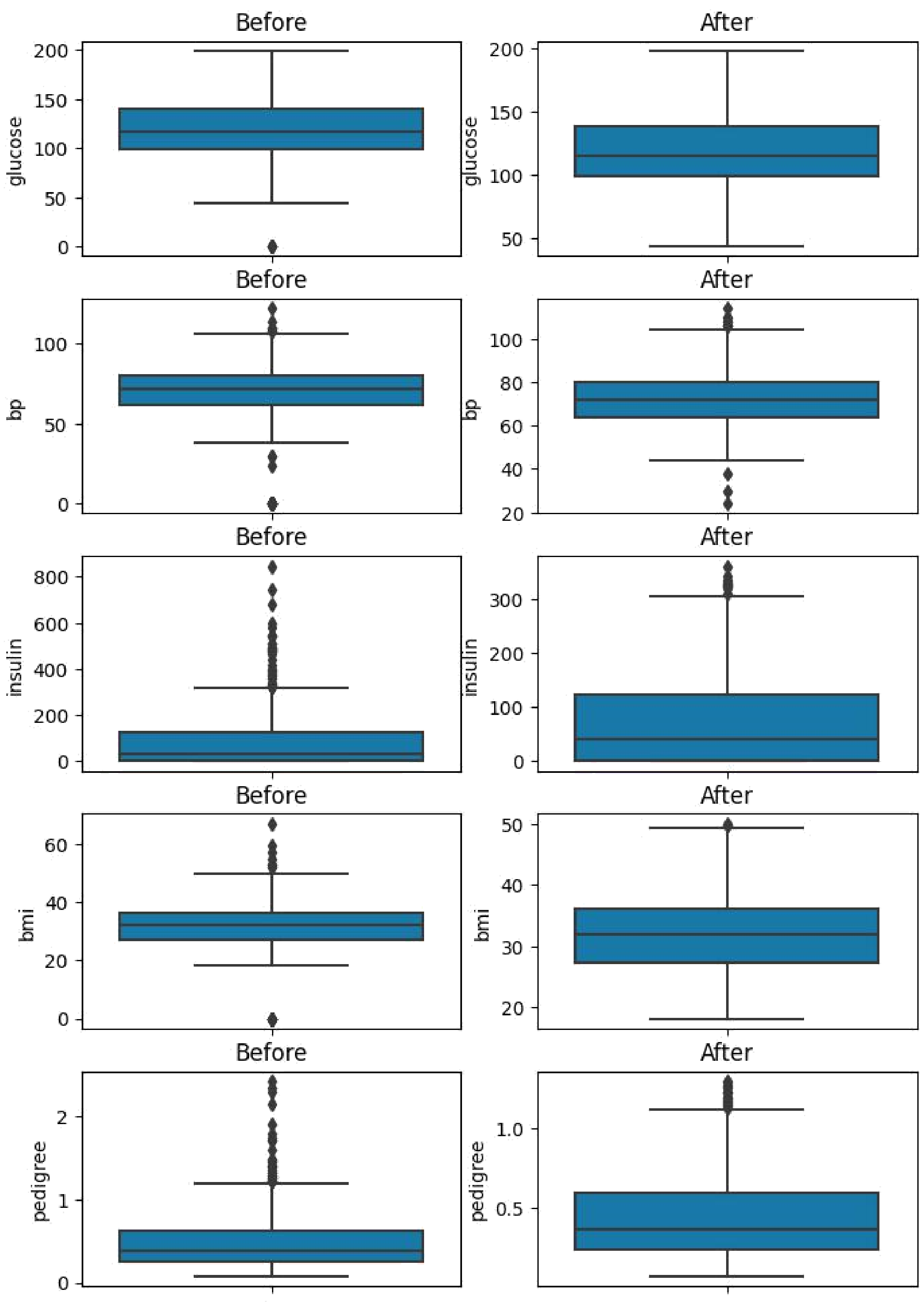
```
[16]:  f,axes = plt.subplots(1,2,figsize=(8,4))
sns.histplot(ax=axes[0],data=X_having_outliers,x="pedigree",kde=True).
 ↪set_title("Before")
sns.histplot(ax=axes[1],data=X,x="pedigree",kde=True).set_title("After")
```

```
[16]: Text(0.5, 1.0, 'After')
```



```
[17]: #comparing boxplots plots after handling the
outliers f,axes = plt.subplots(5,2,figsize=(8,12))

sns.boxplot(ax=axes[0][0],data=X_having_outliers,y="glucose",palette="winter").
 ↪set_title("Before")
sns.boxplot(ax=axes[0][1],data=X,y="glucose",palette="winter").
 ↪set_title("After")


sns.boxplot(ax=axes[1][0],data=X_having_outliers,y="bp",palette="winter").
 ↪set_title("Before")
sns.boxplot(ax=axes[1][1],data=X,y="bp",palette="winter").set_title("After")


sns.boxplot(ax=axes[2][0],data=X_having_outliers,y="insulin",palette="winter").
 ↪set_title("Before")
sns.boxplot(ax=axes[2][1],data=X,y="insulin",palette="winter").
 ↪set_title("After")


sns.boxplot(ax=axes[3][0],data=X_having_outliers,y="bmi",palette="winter").
 ↪set_title("Before")
sns.boxplot(ax=axes[3][1],data=X,y="bmi",palette="winter").set_title("After")
sns.boxplot(ax=axes[4][0],data=X_having_outliers,y="pedigree",palette="winter").
 ↪set_title("Before")
sns.boxplot(ax=axes[4][1],data=X,y="pedigree",palette="winter").
 ↪set_title("After")
```

```
[17]: Text(0.5, 1.0, 'After')
```



From the distribution plots and boxplots shown above, we can see that a lot of outliers have been handled. Now we will use this data to train a Random Forest Classifier model. We will be training a Random Forest Classifier model to determine the principal attributes in the dataset.

2 Random Forest Classifier

Training random forest classifier to determine the principle attributes

[18]:
```
#importing modules for random forest classifier from
sklearn.ensemble import RandomForestClassifier from
sklearn.model_selection import train_test_split from
sklearn.metrics import classification_report from
sklearn.metrics import confusion_matrix

#splitting the dataset into train and test
x_train,x_test,y_train,y_test=train_test_split(X.
 ↪drop('outcome',axis=1),X['outcome'],test_size=0.30,random_state=100)␣
 ↪#without outliers and standardize

print("No. of observations for the training dataset: ",len(x_train))
print("No. of observations for the testing dataset: ",len(x_test))
```

```
No. of observations for the training dataset:  473
No. of observations for the testing dataset:  204
```

[19]:
```
  #making random forest classifier object
rf= RandomForestClassifier(n_jobs=-1,n_estimators=100,random_state=0)

rf.fit(x_train,y_train)  #trining the model
pred=rf.predict(x_test)  #predicting the target attributes for test dataset

acc_rf = rf.score(x_test,y_test)
print("Accuracy of rf is: ",acc_rf,"\n")
```

```
Accuracy of rf is:     0.7843137254901961
```

[20]:
```
#creating a confusion matrix to see actual and predicted results
confusion_df = pd.DataFrame(confusion_matrix(y_test, pred),
                 columns = ["Predicted Outcome " + str(class_name) for class_name␣
 ↪in [0, 1]],
                 index = ["Actual Outcome " + str(class_name) for class_name in␣
 ↪[0, 1]])
confusion_df
```

[20]:
```
                  Predicted Outcome 0 Predicted Outcome 1
Actual Outcome 0                  123                  10
Actual Outcome 1                   34                  37
```

[21]:
```
 col = X.columns #collecting the column names
#view a list of the features & their
importance scores fi = rf.feature_importances_
for i in range(0, len(fi)):
  print(col[i],"=",fi[i])
```

```
glucose = 0.28611949728592045
bp = 0.10406010982541437
insulin = 0.09215271103438498
bmi = 0.18573718776000556
pedigree = 0.1555065645601749
age = 0.17642392953409966
```

As we can see from the calculated feature importance values, bp and insulin contains lower values than the other 4 attributes so we'll eliminate those two and remaining 4 will be our principal attributes

[22]:
```
final = X.drop(["bp","insulin"], axis=1) #dropping bp and insulin
 columns in␣
↪the new dataset
final.describe() #describing the final dataset
```

[22]:
```
          glucose         bmi    pedigree         age    outcome
count  677.000000  677.000000  677.000000  677.000000 677.000000
mean   120.070901   32.128965    0.441594   33.353028   0.327917
std     29.887179    6.457651    0.265363   11.771658   0.469802
min     44.000000   18.200000    0.078000   21.000000   0.000000
25%     99.000000   27.400000    0.244000   24.000000   0.000000
50%    115.000000   32.009138    0.365000   29.000000   0.000000
75%    139.000000   36.200000    0.593000   41.000000   1.000000
max    198.000000   50.000000    1.292000   81.000000   1.000000
```

Saving the modifier dataset containing only principal attributes

[23]:
```
 final.to_csv("/content/final.csv")
```

Splitting the final dataset

[24]:
```
 final_x_train, final_x_test, final_y_train, final_y_test =␣
↪train_test_split(final.drop('outcome',axis=1),final['outcome'],test_size=0.
↪30,random_state=100)

print("Size of final training dataset: ",len(final_x_train))
print("Size of final testing dataset: ",len(final_x_test))
```

```
Size of final training dataset:   473
Size of final testing dataset:    204
```

Standardizing the final dataset

[25]:
```
#standardizing features
from sklearn.preprocessing import StandardScaler

std = StandardScaler()
final_x_std_train = std.fit_transform(final_x_train)
final_x_std_test=std.fit_transform(final_x_test)
```

Splitting dataset which having outliers

[26]:
```
#data with outliers without standardize
X_having_outliers = X_having_outliers.drop(["bp","insulin"], axis=1)
o_x_train, o_x_test, o_y_train, o_y_test = train_test_split(X_having_outliers.
 ↪drop('outcome',axis=1),X_having_outliers['outcome'],test_size=0.
 ↪30,random_state=100)
```

[27]:
```
#data with outliers with standardize
o_x_std_train = std.fit_transform(o_x_train)
o_x_std_test = std.fit_transform(o_x_test)
```

3 Training our required models

Our project goal requires us to train 4 specific classifier models

1. KNN Classifier
2. Naive Bayes Classifier
3. Decision Tree Classifier
4. Logistic Regression

We will be using the dataset obtained after pre-processing the given train dataset to train our required models.

KNN Classifier

[28]:
```
#imporing modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import
GridSearchCV from sklearn.metrics import *
```

[29]:
```
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean', n_jobs=-1)
knn.fit(final_x_std_train, final_y_train)

pipe = Pipeline([('standardizer', std), ('knn', knn)])


search_space = [{'knn__n_neighbors':[1,2,3,4,5,6,7,8,9,11,13]}]
 #create space␣
 ↪of candidate values
```

```
clf = GridSearchCV(pipe, search_space, cv=5,

 verbose=1).fit(final_x_std_train,␣
↪final_y_train) #grid search
print("k = ",clf.best_estimator_.get_params()["knn__n_neighbors"])
```

```
Fitting 5 folds for each of 11 candidates, totalling
55 fits k = 7
```

We standardize the data. The reason is that the value of k remains 7.

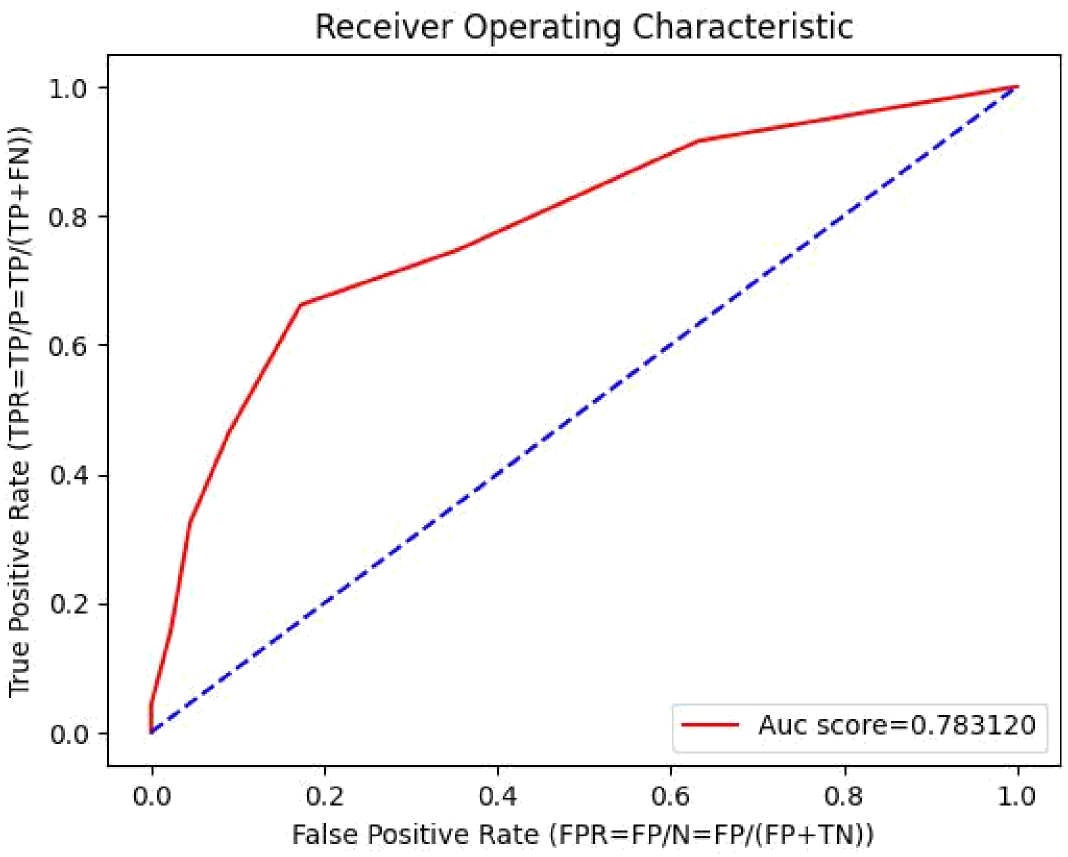
[30]:
```
 #Training knn with K as 7
knn2 = KNeighborsClassifier(n_neighbors=7, metric='euclidean', n_jobs=-1)
knn2.fit(final_x_std_train, final_y_train)
#model has been trained
pred_knn2=knn2.predict(final_x_std_test)
```

[31]:
```
 #create confusino matrix
confusion_df = pd.DataFrame(confusion_matrix(final_y_test, pred_knn2),
                  columns = ["Predicted Class " + str(class_name) for class_name in␣

 ↪[0, 1]],  index = ["Actual Class " + str(class_name) for class_name in [0,␣
 ↪ 1]])
confusion_df
```

[31]:
```
                  Predicted Class 0  Predicted Class 1
Actual Class 0                  121                 12
Actual Class 1                   38                 33
```

[32]:
```
 #ROC graph for K-NN
y_score=knn2.predict_proba(final_x_std_test)[:,1]

fpr,tpr,threshold=roc_curve(final_y_test,y_score)
roc_auc=auc(fpr,tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr,tpr,'r',label="Auc score=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],'b',ls="--")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```

[32]:
```
Text(0.5, 0, 'False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```



From the Reciever Operating Characteristic (ROC) graph, we find the Area Under Curve(AUC) for our k-NN classifier model is 0.783120

AUC value of k-NN = 0.783120

[33]:
```
 acc_knn = knn2.score(final_x_std_test,final_y_test)
print("Accuracy of knn #knn without outliers and with standardize:␣
 ↪",acc_knn,"\n")
```

```
Accuracy of knn #knn without outliers and with standardize:
                                                    0.7549019607843137
```

[34]:
```
 #knn with outliers and without standardize

knn3 = KNeighborsClassifier(n_neighbors=7, metric='euclidean',

 n_jobs=-1). ↪fit(o_x_train, o_y_train)
acc_knn3 = knn3.score(o_x_test,o_y_test)
print("Accuracy of knn with outliers and without standardize: ",acc_knn3,"\n")
```

```
Accuracy of knn with outliers and without standardize: 0.7142857142857143
```

```
[35]:  #knn with outliers and with standardize
knn4 = KNeighborsClassifier(n_neighbors=7, metric='euclidean',
 n_jobs=-1). ↪fit(o_x_std_train, o_y_train)
acc_knn4 = knn4.score(o_x_std_test,o_y_test)
print("Accuracy of knn with outliers and with standardize: ",acc_knn4,"\n")
```

```
Accuracy of knn with outliers and with standardize:    0.7186147186147186
```

```
[36]:  #knn without outliers and without standardize

knn5 = KNeighborsClassifier(n_neighbors=7, metric='euclidean',

 n_jobs=-1). ↪fit(final_x_train, final_y_train)
acc_knn5 = knn5.score(final_x_test,final_y_test)

print("Accuracy of knn without outliers and without
 standardize:␣ ↪",acc_knn5,"\n")
```

```
Accuracy of knn without outliers and without standardize: 0.75
```

Naive Bayes

We will now train a Gaussian Naive Bayes classifier model using our dataset

```
[37]:  #importing Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

```
[38]:  nb=GaussianNB()
nb.fit(final_x_train,final_y_train)

#naive bayes without outliers
pred_nb=nb.predict(final_x_test)

acc_nb = nb.score(final_x_test,final_y_test)
print("Accuracy of nb is: ",acc_nb)
```

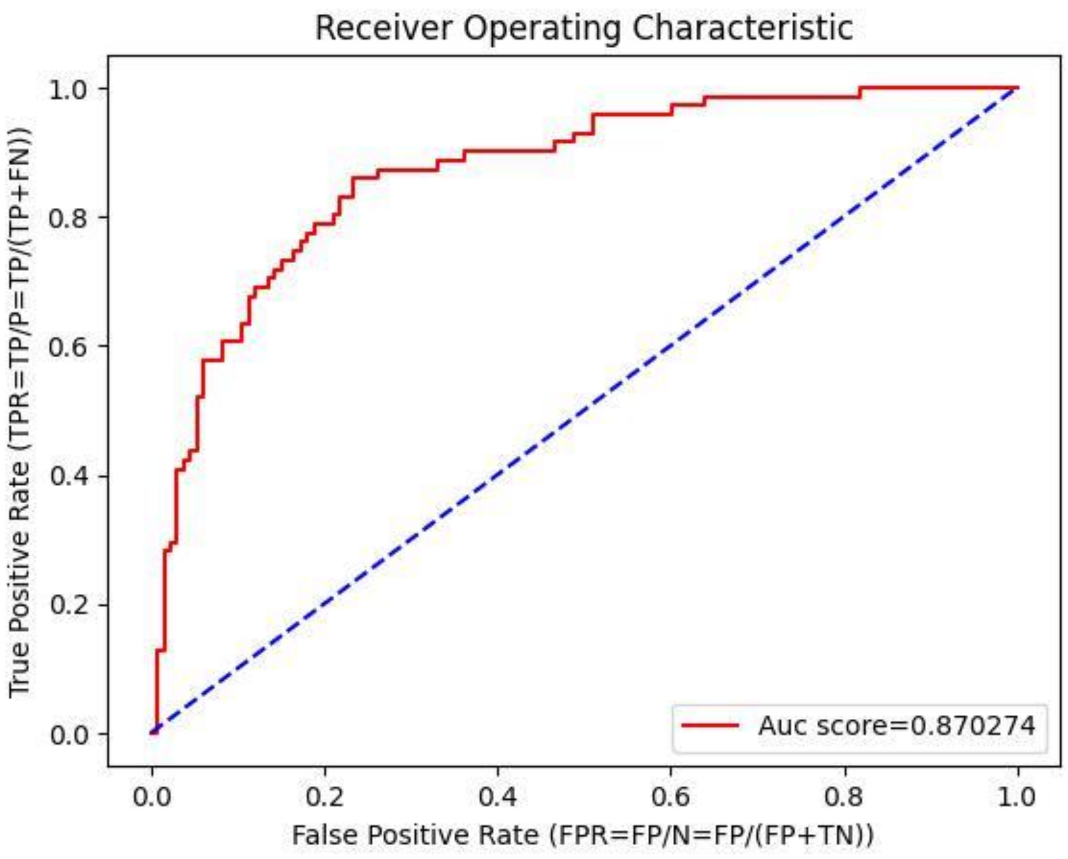
```
Accuracy of nb is:  0.7941176470588235
```

```
[39]:  confusion_df = pd.DataFrame(confusion_matrix(y_test, pred_nb),
             columns = ["Predicted Class " + str(class_name) for class_name in␣

 ↪[0, 1]],  index = ["Class " + str(class_name) for class_name in [0, 1]])
confusion_df
```

```
[39]:         Predicted Class 0  Predicted Class 1
Class 0                125                  8
Class 1                 34                 37
```

```
[40]:  #ROC graph for Naive Bayes
y_score=nb.predict_proba(final_x_test)[:,1] #this line returns probability for␣
 ↪each row of the dataset, [:,1] for 1 probability
fpr1,tpr1,threshold1=roc_curve(final_y_test,y_score)
roc_auc=auc(fpr1,tpr1)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr1,tpr1,'r',label="Auc score=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],'b',ls="--")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```

```
[40]: Text(0.5, 0, 'False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```



From the Reciever Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Gaussian Naive Bayes classifier model is 0.870274

AUC vaues of Naive Bayes = 0.870274

[41]:
```
 #naive bayes with outliers
nb1=GaussianNB()
nb1.fit(final_x_train,final_y_train)
nb1.fit(o_x_train,o_y_train)

acc_nb1 = nb1.score(o_x_test,o_y_test)
print("Accuracy of naive bayes with outliers: ",acc_nb1,"\n")
```

```
Accuracy of naive bayes with outliers: 0.7445887445887446
```

Decision Tree

[42]:
```
 #importing Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
```

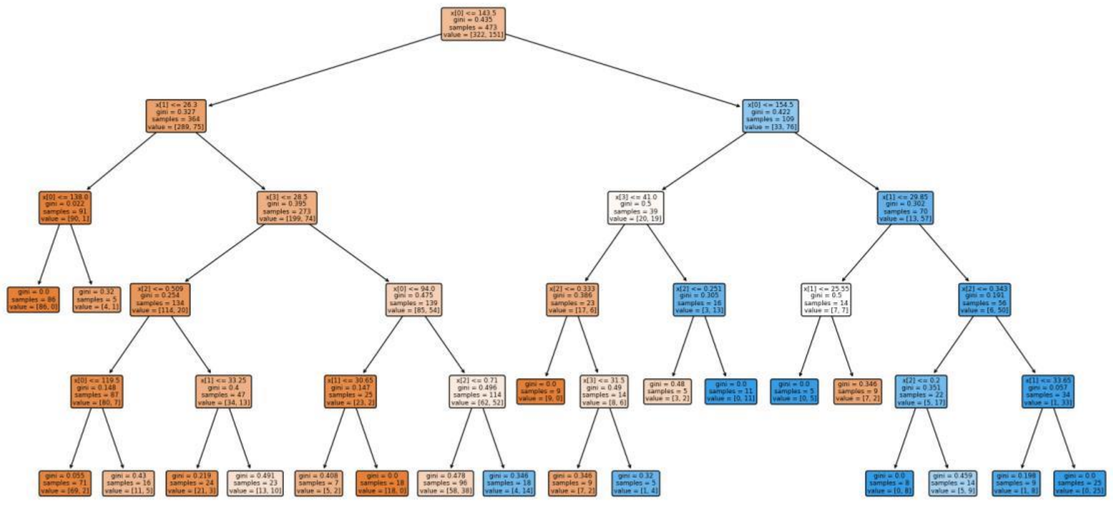
[43]:
```
 dtree=DecisionTreeClassifier(min_samples_leaf=5,max_depth=5,random_state=100)

#decision tree without outliers
dtree.fit(final_x_train,final_y_train)
pred_dt=dtree.predict(final_x_test)
acc_dt = dtree.score(final_x_test,final_y_test)
print("Accuracy of decision tree without outliers without
  standardize:␣ ",acc_dt,"\n")
```

```
Accuracy of decision tree without outliers without standardize:
0.7549019607843137
```

[44]:
```
 #to draw the Decision Tree
from sklearn import tree
with open("FinalDTree.txt","w") as a:
  a = tree.export_graphviz(dtree, out_file = a)
```

[45]:
```
 from sklearn import tree
import matplotlib.pyplot as plt
plt.subplots(figsize = (20, 10)) # for resizing the graph
tree.plot_tree(dtree, filled = True, rounded = True)
```

[45]:
```
[Text(0.425, 0.9166666666666666, 'x[0] <= 143.5\ngini =
0.435\nsamples = 473\nvalue = [322, 151]'),
 Text(0.15714285714285714, 0.75, 'x[1] <= 26.3\ngini =
0.327\nsamples = 364\nvalue = [289, 75]'),
 Text(0.05714285714285714, 0.5833333333333334, 'x[0] <=
138.0\ngini = 0.022\nsamples = 91\nvalue = [90, 1]'),
 Text(0.02857142857142857, 0.4166666666666667, 'gini = 0.0\nsamples
= 86\nvalue = [86, 0]'),
 Text(0.08571428571428572, 0.4166666666666667, 'gini = 0.32\nsamples = 5\nvalue
= [4, 1]'),
 Text(0.2571428571428571, 0.5833333333333334, 'x[3] <=
28.5\ngini = 0.395\nsamples = 273\nvalue = [199, 74]'),
 Text(0.14285714285714285, 0.4166666666666667, 'x[2] <=
0.509\ngini = 0.254\nsamples = 134\nvalue = [114, 20]'),
 Text(0.08571428571428572, 0.25, 'x[0] <= 119.5\ngini =
0.148\nsamples = 87\nvalue = [80, 7]'),
 Text(0.05714285714285714, 0.08333333333333333, 'gini =
0.055\nsamples = 71\nvalue = [69, 2]'),
 Text(0.11428571428571428, 0.08333333333333333, 'gini =
0.43\nsamples = 16\nvalue = [11, 5]'),
 Text(0.2, 0.25, 'x[1] <= 33.25\ngini = 0.4\nsamples = 47\nvalue = [34, 13]'),
 Text(0.17142857142857143, 0.08333333333333333, 'gini = 0.219\nsamples =
24\nvalue = [21, 3]'),
 Text(0.22857142857142856, 0.08333333333333333, 'gini =
0.491\nsamples = 23\nvalue = [13, 10]'),
 Text(0.37142857142857144, 0.4166666666666667, 'x[0] <=
94.0\ngini = 0.475\nsamples = 139\nvalue = [85, 54]'),
 Text(0.3142857142857143, 0.25, 'x[1] <= 30.65\ngini =
0.147\nsamples = 25\nvalue = [23, 2]'),
 Text(0.2857142857142857, 0.08333333333333333, 'gini = 0.408\nsamples
= 7\nvalue = [5, 2]'),
 Text(0.34285714285714286, 0.08333333333333333, 'gini = 0.0\nsamples
= 18\nvalue = [18, 0]'),
 Text(0.42857142857142855, 0.25, 'x[2] <= 0.71\ngini =
0.496\nsamples = 114\nvalue = [62, 52]'),
 Text(0.4, 0.08333333333333333, 'gini = 0.478\nsamples = 96\nvalue = [58, 38]'),
 Text(0.45714285714285713, 0.08333333333333333, 'gini = 0.346\nsamples =
18\nvalue = [4, 14]'),
 Text(0.6928571428571428, 0.75, 'x[0] <= 154.5\ngini =
0.422\nsamples = 109\nvalue = [33, 76]'),
 Text(0.5714285714285714, 0.5833333333333334, 'x[3] <= 41.0\ngini =
0.5\nsamples = 39\nvalue = [20, 19]'),
 Text(0.5142857142857142, 0.4166666666666667, 'x[2] <=
0.333\ngini = 0.386\nsamples = 23\nvalue = [17, 6]'),
 Text(0.4857142857142857, 0.25, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.5428571428571428, 0.25, 'x[3] <= 31.5\ngini = 0.49\nsamples = 14\nvalue
= [8, 6]'),
 Text(0.5142857142857142, 0.08333333333333333, 'gini = 0.346\nsamples
= 9\nvalue = [7, 2]'),
 Text(0.5714285714285714, 0.08333333333333333, 'gini = 0.32\nsamples
= 5\nvalue = [1, 4]'),
 Text(0.6285714285714286, 0.4166666666666667, 'x[2] <=
0.251\ngini = 0.305\nsamples = 16\nvalue = [3, 13]'),
 Text(0.6, 0.25, 'gini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.6571428571428571, 0.25, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.8142857142857143, 0.5833333333333334, 'x[1] <= 29.85\ngini =
0.302\nsamples = 70\nvalue = [13, 57]'),
 Text(0.7428571428571429, 0.4166666666666667, 'x[1] <= 25.55\ngini =
0.5\nsamples = 14\nvalue = [7, 7]'),
 Text(0.7142857142857143, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.7714285714285715, 0.25, 'gini = 0.346\nsamples = 9\nvalue = [7, 2]'),
 Text(0.8857142857142857, 0.4166666666666667, 'x[2] <= 0.343\ngini =
0.191\nsamples = 56\nvalue = [6, 50]'),
 Text(0.8285714285714286, 0.25, 'x[2] <= 0.2\ngini = 0.351\nsamples = 22\nvalue
= [5, 17]'),
 Text(0.8, 0.08333333333333333, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
 Text(0.8571428571428571, 0.08333333333333333, 'gini = 0.459\nsamples =
14\nvalue = [5, 9]'),
 Text(0.9428571428571428, 0.25, 'x[1] <= 33.65\ngini = 0.057\nsamples =
34\nvalue = [1, 33]'),
 Text(0.9142857142857143, 0.08333333333333333, 'gini = 0.198\nsamples = 9\nvalue
= [1, 8]'),
 Text(0.9714285714285714, 0.08333333333333333, 'gini = 0.0\nsamples = 25\nvalue
= [0, 25]')]
```

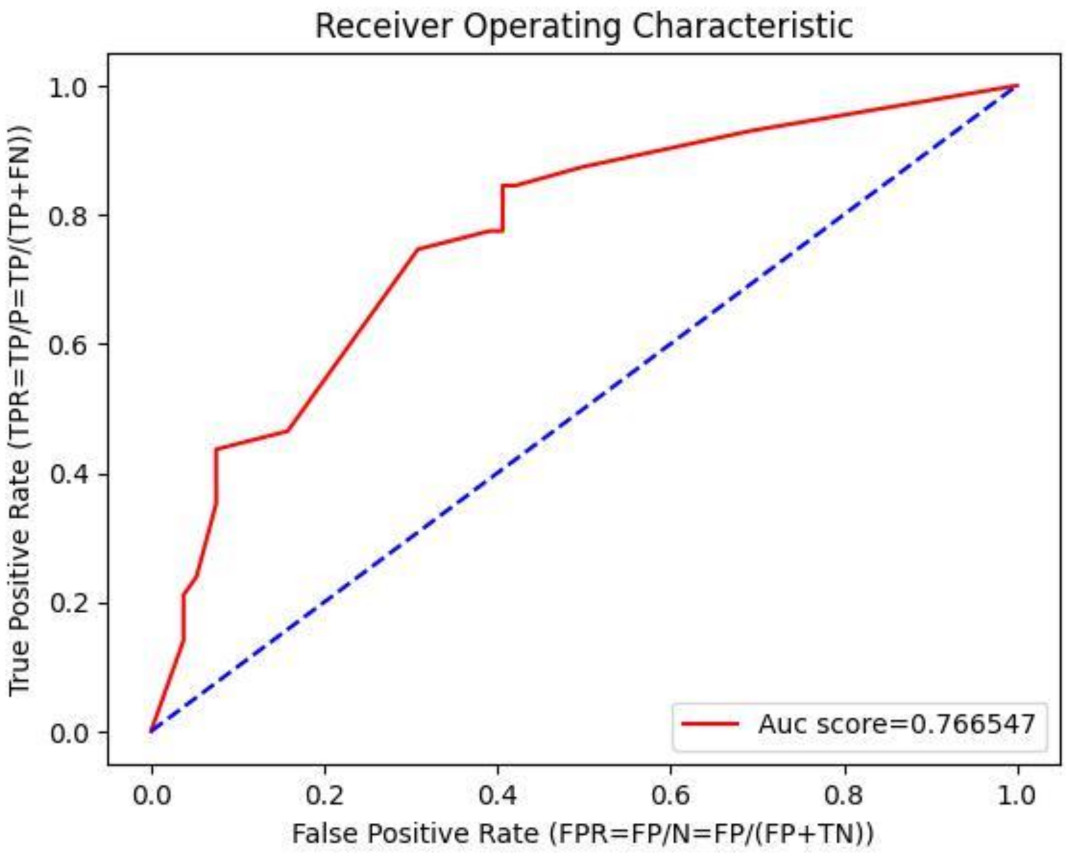


```
[46]: pd.DataFrame(confusion_matrix(y_test, pred_dt),
                   columns = ["Predicted Class " + str(class_name) for class_name in␣
       ↪[0, 1]],  index = ["Class " + str(class_name) for class_name in [0, 1]])
```

```
[46]:          Predicted Class 0  Predicted Class 1
      Class 0                123                 10
      Class 1                 40                 31
```

```
[47]:  #ROC graph for Decision Tree
y_score=dtree.predict_proba(final_x_test)[:,1]
fpr2,tpr2,threshold2=roc_curve(final_y_test,y_score)
roc_auc=auc(fpr2,tpr2)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr2,tpr2,'r',label="Auc score=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],'b',ls="--")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```

```
[47]: Text(0.5, 0, 'False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```



From the Reciever Operating Characteristic(ROC) graph, we find the Area Under Curve(AUC) for our Decision Tree classifier model is 0.766547

AUC value of Decision Tree = 0.766547

```
[48]:  #decision tree with outliers
dtree1 = DecisionTreeClassifier(min_samples_leaf=5,max_depth=5,random_state=100)
```

```
dtree1.fit(o_x_train,o_y_train)

acc_dtree1 = dtree1.score(o_x_test,o_y_test)
print("Accuracy of decision tree with outliers: ",acc_dtree1,"\n")
```

```
Accuracy of decision tree with outliers:     0.6623376623376623
```

Logistic Regression Model

[49]:
```
 #importing required modules
from sklearn.linear_model import LogisticRegression
```

[50]:
```
logmodel = LogisticRegression()

#logistic regression without outliers
logmodel.fit(final_x_train,final_y_train)
pred_log = logmodel.predict(final_x_test)
acc_lr = logmodel.score(final_x_test,final_y_test)
print("Accuracy of logistic regression is withour outliers: ",acc_lr,"\n")
```

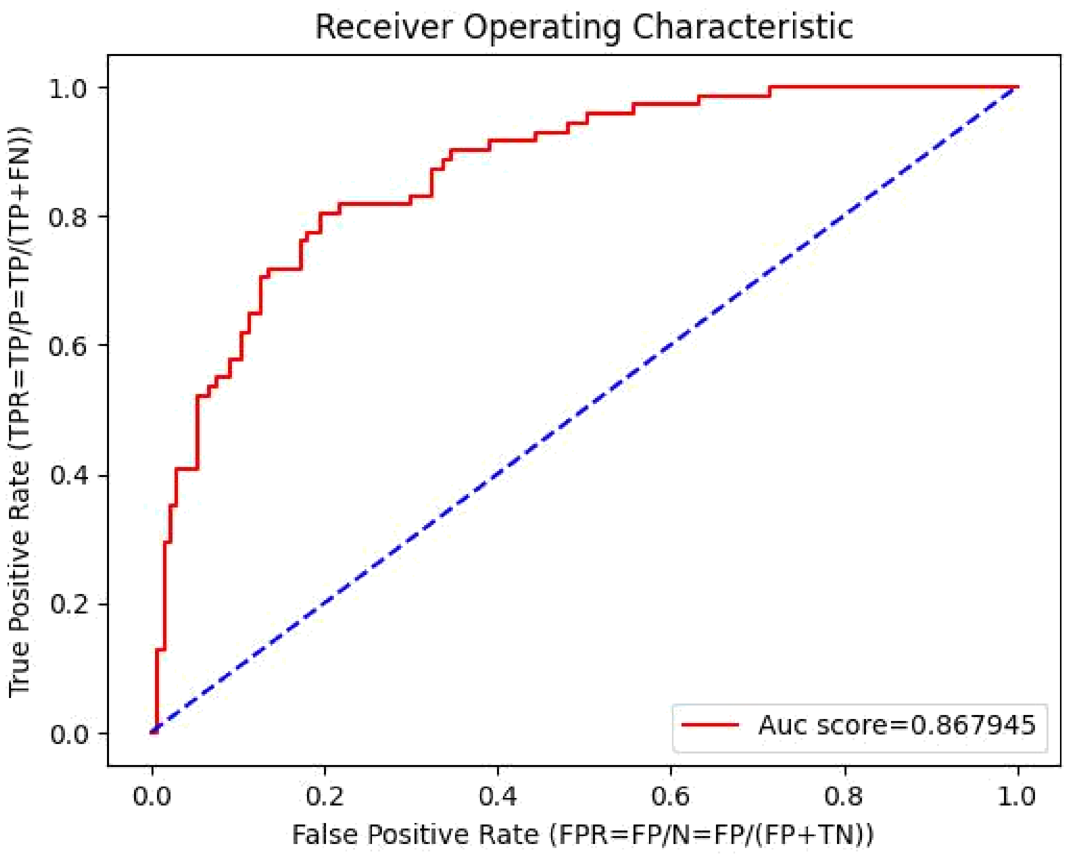
```
Accuracy of logistic regression is withour outliers:   0.7794117647058824
```

[51]:
```
 #Create confusion matrix
pd.DataFrame(confusion_matrix(final_y_test, pred_log),
  columns = ["Predicted Class " + str(class_name) for class_name in␣
 ↪[0, 1]],
               index = ["Class " + str(class_name) for class_name in [0, 1]])
```

[51]:
```
         Predicted Class 0   Predicted Class 1
Class 0                126                   7
Class 1                 38                  33
```

[52]:
```
 #ROC graph Logistic Regression
y_score=logmodel.predict_proba(final_x_test)[:,1]
fpr3,tpr3,threshold3=roc_curve(final_y_test,y_score)
roc_auc=auc(fpr3,tpr3)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr3,tpr3,'r',label="Auc score=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],'b',ls="--")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```

[52]:
```
Text(0.5, 0, 'False Positive Rate (FPR=FP/N=FP/(FP+TN))')
```



From the Reciever operating Characteristic(ROC) graph, we find the Area Under Curve(AUC) for our Logistic Regression classifier model is 0.867945

[53]:
```
#printing the coefficients of
logmodel fcol = final_x_test.columns

lm = np.asarray(logmodel.coef_).flatten()
for i in range(0, len(lm)):
  print(fcol[i],"=",lm[i])
```

```
glucose = 0.03671779767068327
bmi = 0.07746382582511345
pedigree = 1.1764318757470082
age = 0.024738275530206234
```

[54]:
```
  #R Square for logistic regression
from sklearn.metrics import mean_squared_error,r2_score

print('R2_Score:',r2_score(final_y_test,pred_log),"\n")
```

```
R2_Score: 0.027851318436937555
```

```
[55]:  #RMSE for logistic regression
       mse=mean_squared_error(final_y_test,pred_log)
       rmse=np.sqrt(mse)
       print("mean_squared_error is %f and rmse is " %mse ,rmse)
```

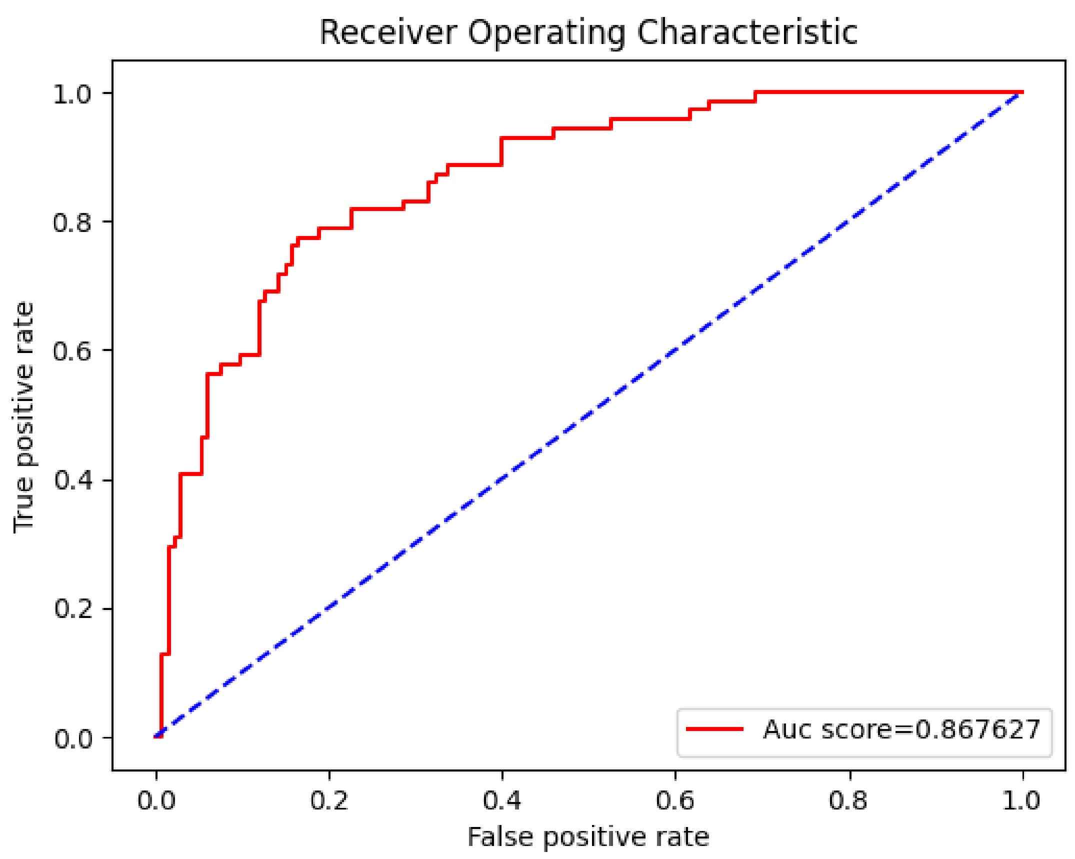
```
mean_squared_error is 0.220588 and rmse is  0.46966821831386213
```

```
[56]:  #logistic regression without outliers and with standardize
       logmodel2=LogisticRegression()
       logmodel2.fit(final_x_std_train,final_y_train)

       acc_log2 = logmodel2.score(final_x_std_test,final_y_test)
       print("Accuracy of logistic regression with standardize: ",acc_log2,"\n")
```

```
Accuracy of logistic regression with standardize: 0.7794117647058824
```

```
[57]: y_score=logmodel2.predict_proba(final_x_std_test)[:,1]
      fpr3,tpr3,threshold3=roc_curve(final_y_test,y_score)
      roc_auc=auc(fpr3,tpr3)
      plt.title('Receiver Operating Characteristic')
      plt.plot(fpr3,tpr3,'r',label="Auc score=%f"%roc_auc)
      plt.legend(loc='lower right')
      plt.plot([0,1],'b',ls="--")
      plt.ylabel('True positive rate')
      plt.xlabel('False positive rate')
```

```
[57]: Text(0.5, 0, 'False positive rate')
```



```
[58]:  #logistic regression with outliers and without standardize
logmodel3=LogisticRegression()
logmodel3.fit(o_x_train,o_y_train)

acc_log3 = logmodel3.score(o_x_test,o_y_test)

print("Accuracy of logistic regression with outlier, without
  standardize:␣ ",acc_log3,"\n")
```

```
Accuracy of logistic regression with outlier, without standardize:
0.7402597402597403
```

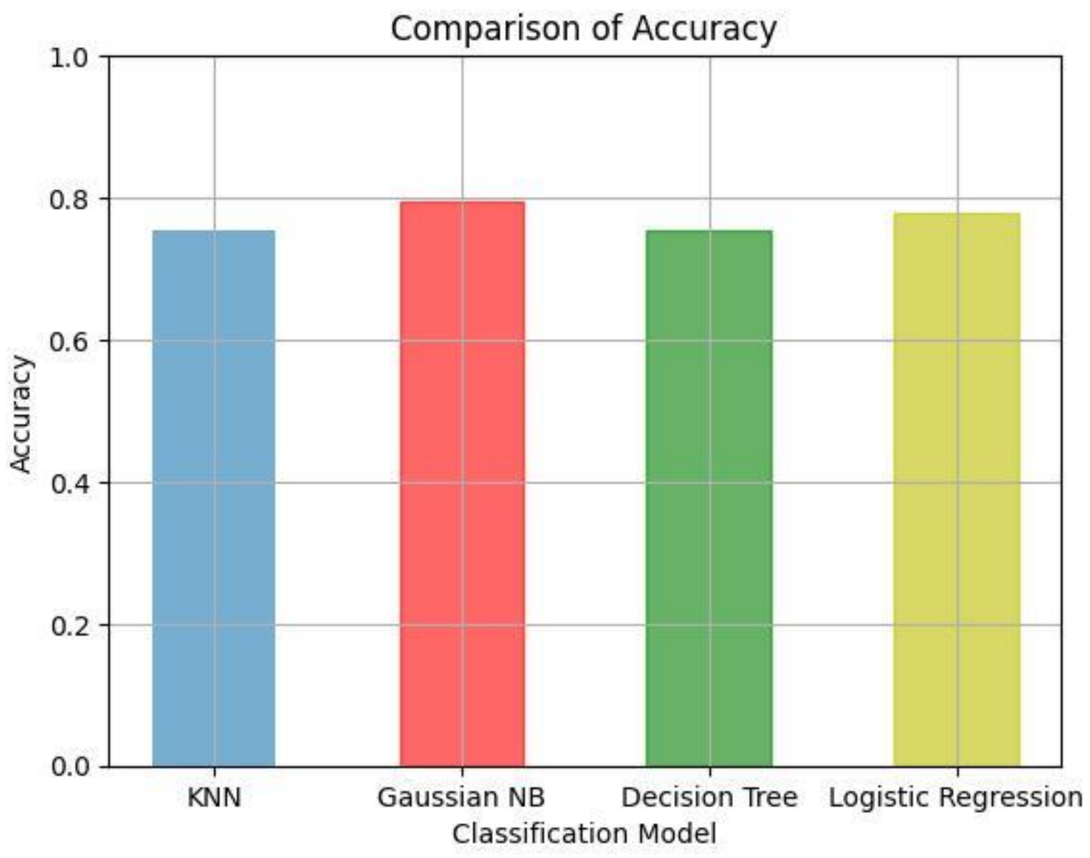
Comparing accuracy of all 4 models

```
[59]:  #printing the accuracy scores of all the 4 models:
print("KNN: ",acc_knn)
print("GNB: ",acc_nb)
print("LR: ",acc_lr)
print("DTree: ",acc_dt)
```

```
KNN: 0.7549019607843137
```

```
GNB: 0.7941176470588235
LR:  0.7794117647058824
DTree: 0.7549019607843137
```

```
[60]:  #comparing accuracies of 4 models using barplot
models = ["KNN", "Gaussian NB", "Decision Tree", "Logistic
Regression"] accuracies = [acc_knn, acc_nb, acc_dt, acc_lr]
barlist = plt.bar(models, accuracies, width=0.5, alpha=0.6)
plt.ylim(0,1.0)
barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('y')
plt.xlabel('Classification Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Accuracy')
plt.grid()
```



Saving Gaussian Naive Bayes model using pickle

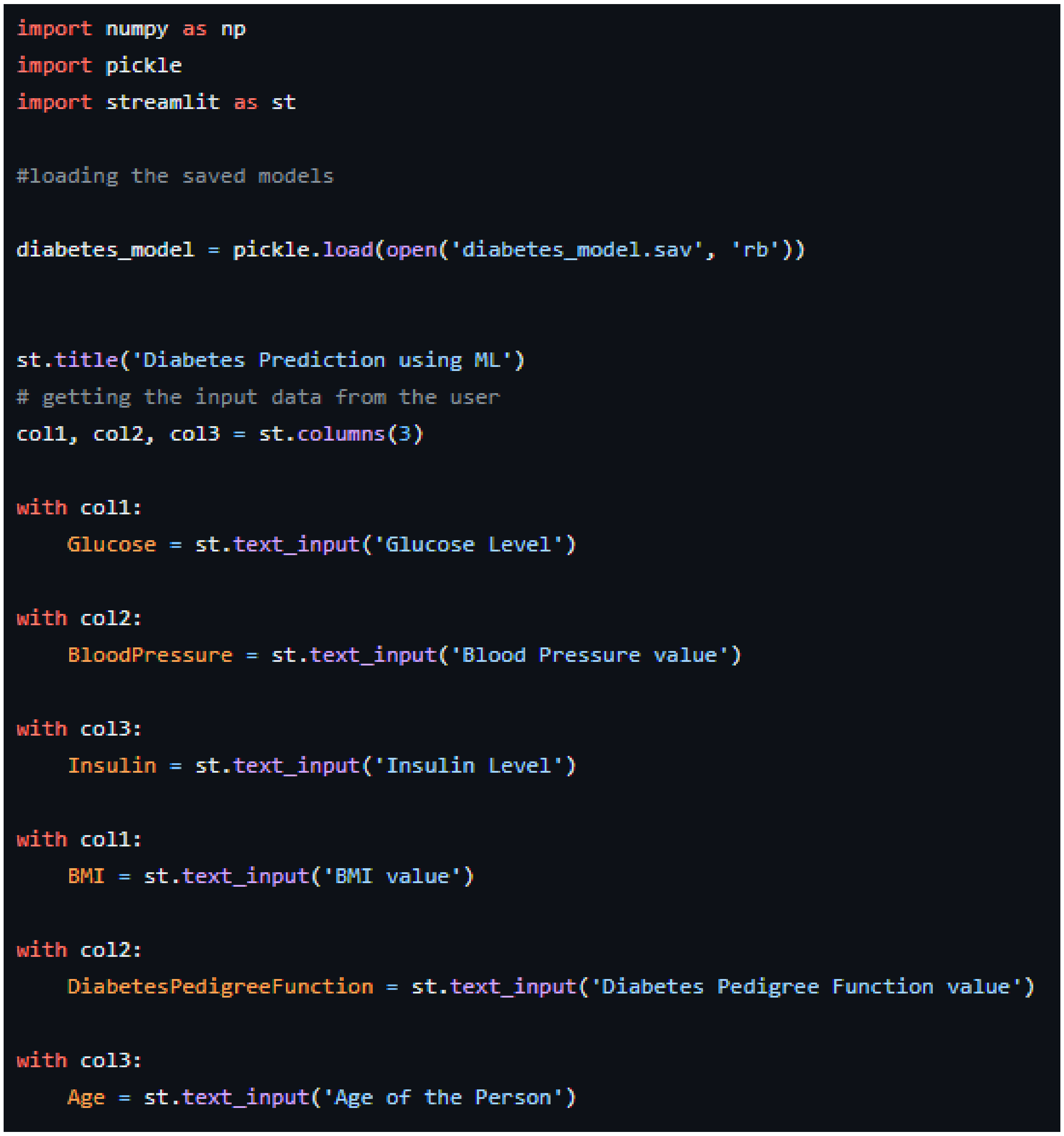
[61]:
```
 import pickle
filename = 'diabetes_model.sav'
pickle.dump(nb, open(filename, 'wb'))

# loading the saved model
loaded_model = pickle.load(open('diabetes_model.sav', 'rb'))

input_data = (166,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = loaded_model.predict(input_data_reshaped)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[1]
The person is diabetic
```

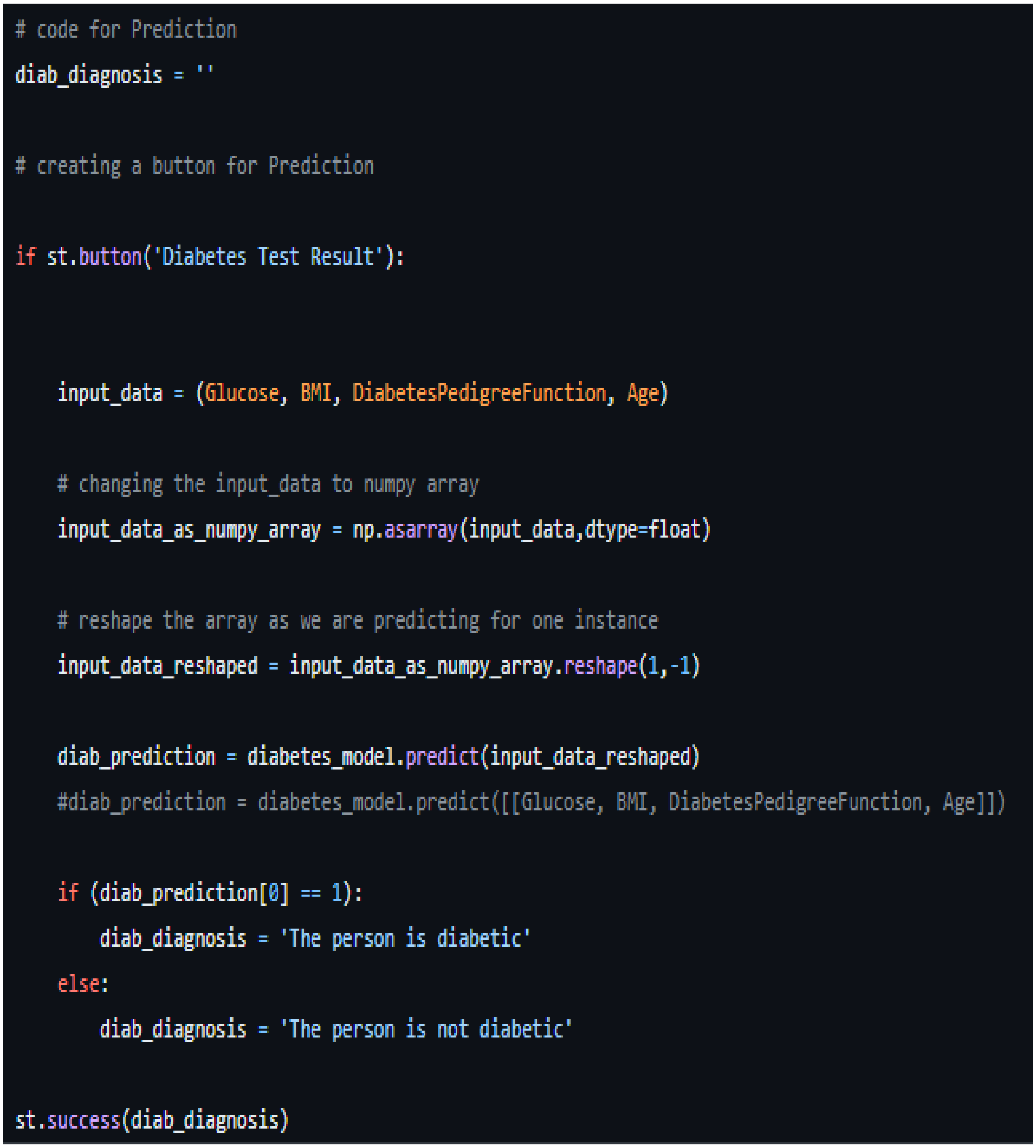
User Interface

To build the user interface for our project we used streamlit. Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers.

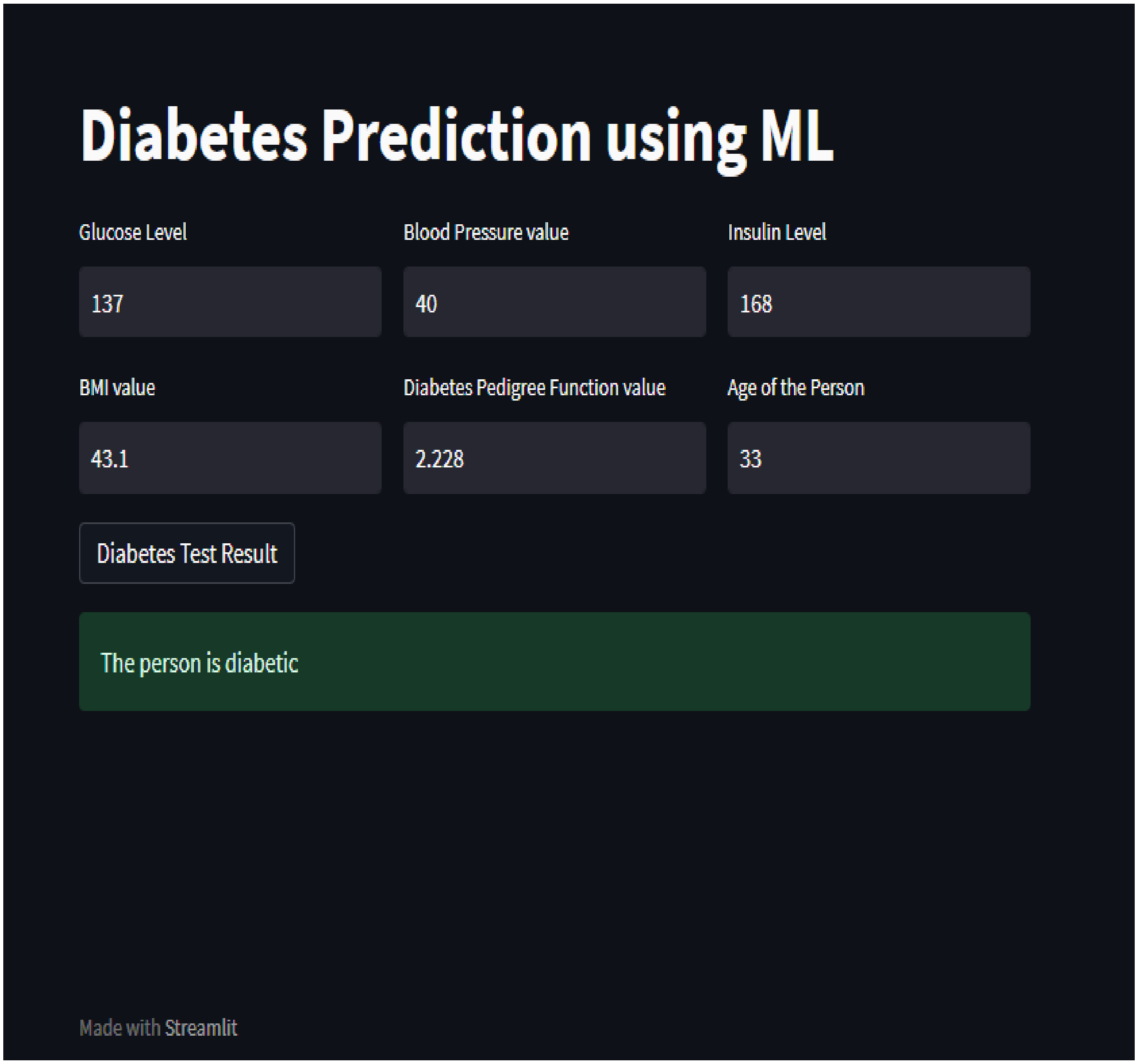
CODE:



```python
import numpy as np
import pickle
import streamlit as st

#loading the saved models

diabetes_model = pickle.load(open('diabetes_model.sav', 'rb'))


st.title('Diabetes Prediction using ML')
# getting the input data from the user
col1, col2, col3 = st.columns(3)

with col1:
    Glucose = st.text_input('Glucose Level')

with col2:
    BloodPressure = st.text_input('Blood Pressure value')

with col3:
    Insulin = st.text_input('Insulin Level')

with col1:
    BMI = st.text_input('BMI value')

with col2:
    DiabetesPedigreeFunction = st.text_input('Diabetes Pedigree Function value')

with col3:
    Age = st.text_input('Age of the Person')
```



```python
# code for Prediction
diab_diagnosis = ''

# creating a button for Prediction

if st.button('Diabetes Test Result'):



    input_data = (Glucose, BMI, DiabetesPedigreeFunction, Age)

    # changing the input_data to numpy array
    input_data_as_numpy_array = np.asarray(input_data,dtype=float)

    # reshape the array as we are predicting for one instance
    input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

    diab_prediction = diabetes_model.predict(input_data_reshaped)
    #diab_prediction = diabetes_model.predict([[Glucose, BMI, DiabetesPedigreeFunction, Age]])

    if (diab_prediction[0] == 1):
        diab_diagnosis = 'The person is diabetic'
    else:
        diab_diagnosis = 'The person is not diabetic'

st.success(diab_diagnosis)
```



Diabetes Prediction using ML

Glucose Level: 137

Blood Pressure value: 40

Insulin Level: 168

BMI value: 43.1

Diabetes Pedigree Function value: 2.228

Age of the Person: 33

The person is diabetic

https://subhajit-ghatak-diabetes-prediction-ml-webapp-prediction-p27639.streamlit.app/

Future Scope of Improvements

- Various hospital institutions can use these models and modify them according to their needs to use in predicting the diabetes of their patients. This will reduce the manual labour and time spent on determining whether the patient is diabetic or not.
- Patients who intend to know about their diabetic status can use these trained models to check whether they are diabetic or not. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- Correlation and feature importance of "blood pressure" and "insulin" is very low. With more data and further analysis, it might be possible to describe the reason.

Certificate

This is to certify that Mr. Subhajit Ghatak of Asansol Engineering College, Registration number: 211080571010015, has successfully completed a project on Diabetes Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that Mr. Saurabh Thakur of Asansol Engineering College, Registration number: 211080571010011, has successfully completed a project on Diabetes Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

[Name of your faculty]

Asansol Engineering College

Certificate

This is to certify that Mr. Ashish Dungdung of Asansol Engineering College, Registration number: 211080571010020, has successfully completed a project on Diabetes Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

[Name of your faculty]
Asansol Engineering College

Certificate

This is to certify that Mr. Charanjit Singh of Asansol Engineering College, Registration number: 211080571010034, has successfully completed a project on Diabetes Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

[Name of your faculty]
Asansol Engineering College

Certificate

This is to certify that Mr. Smriti Sengupta of Asansol Engineering College, Registration number: 211080571010058, has successfully completed a project on Diabetes Prediction using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

[Name of your faculty]

Asansol Engineering College

Department of Computer Applications

Asansol Engineering College

Kalyanpur, Sen Raleigh Road, Asansol – 713304

CERTIFICATE

*This is to certify that the project work entitled “…**Diabetes Prediction**……” using machine learning with python is a bonafide record of work carried out in the **Department of Computer Application**, Asansol Engineering College, Asansol.*

Name	*Roll Number*	*Registration Number*
Subhajit Ghatak	10871021025	211080571010015
Saurabh Thakur	10871021027	211080571010011
Ashish Dungdung	10871021053	211080571010020
Charanjit Singh	10871021033	211080571010034
Smriti Sengupta	10871021052	211080571010058

The students of 4th Semester MCA 2022-23 under the guidance of Prof. Arnab Chakraborty in requirement of fulfillment of the Award of Degree of MCA from Maulana Abul Kalam Azad University of Technology, West Bengal.

Signature of
The Project Guide
Name: Dr./Mr…………………………..
Designation: Assistant Professor

Recommendation & Signature of
The Principal
Name: Dr. P. P. Bhattacharya

Recommendation &Signature of
The Head of Department
Name: Dr. P. PAL

Recommendation &Signature of
Internal / External Examiners