PROJECT DESIGN PHASE - 5
AI BASED DIABETES PREDICTION SYSTEM

INTRODUCTION:

The accuracy level was 90% using the random forest algorithm, which is much higher when compared to other algorithms.

In a recent paper [5], Mohan and Jain used the SVM algorithm to analyze and predict diabetes with the help of the Pima Indian Diabetes Dataset.

Our proposed model outperforms other machine learning models, including k-NN, SVM, DT, RF, AdaBoost, and GNB, in predicting diabetes.

The model achieves high average accuracy, precision, recall, F1-score, and AUC values of 0.9887, 0.9861, 0.9792, 0.9851, and 0.999, respectively.

The AI model was trained on over 270,000 X-ray images from 160,000 patients, with deep learning determining the image features that best predicted a
later diagnosis of diabetes.

A system is used to predict whether a patient has diabetes based on some ofits
health-related details such as BMI (Body Mass Index), blood pressure, Insulin, etc.

AI in Healthcare is an industry that always makes it necessary to make a precise
decision, whether it is a treatment, test, or discharge. Diabetes is common due to
modern food intake, and it is necessary to keep track of the body. AI in Diabetes helps
to predict or Detect Diabetes. Any neglect in health can have a high cost for the patients
and the medical practitioner. It becomes challenging for the patient to trust that this
decision is taken by the machine that does not explain how it reaches a particular
conclusion.

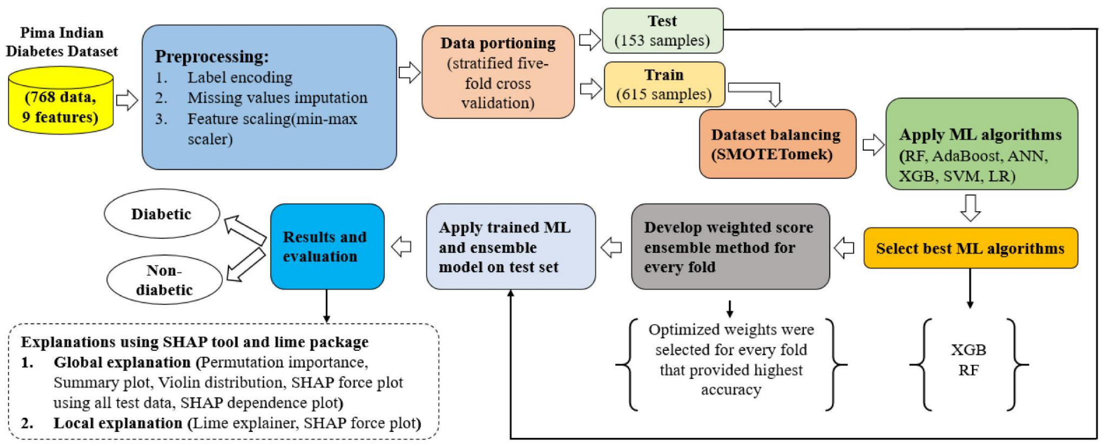
METHODOLOGY:

The whole workflow of the proposed approach is demonstrated in Figure 1. The data was downloaded from Kaggle (https://www.kaggle.com/datasets/mathchi/diabetes-data-set) then it was cleaned and pre-processed (missing values imputation, class balance, etc.).

Diabetes related diseases have recently become one of the top ten causes of death in developing countries.

The government and individuals are funding research projects to find an easier and faster way to detect the disease at an early stage.

There are two types of diabetes: type-1 and type-2. Type 2 diabetes is characterized by high blood sugar, insulin resistance, and a relative lack of insulin. Insulin resistance occurs due to


Pima Indian Diabetes Dataset
(768 data, 9 features)
Preprocessing:
1. Label encoding
2. Missing values imputation
3. Feature scaling(min-max scaler)
Data portioning (stratified five-fold cross validation)
Test (153 samples)
Train (615 samples)
Dataset balancing (SMOTETomek)
Apply ML algorithms (RF, AdaBoost, ANN, XGB, SVM, LR)
Select best ML algorithms
XGB RF
Develop weighted score ensemble method for every fold
Optimized weights were selected for every fold that provided highest accuracy
Apply trained ML and ensemble model on test set
Results and evaluation
Diabetic
Non-diabetic
Explanations using SHAP tool and lime package
1. Global explanation (Permutation importance, Summary plot, Violin distribution, SHAP force plot using all test data, SHAP dependence plot)
2. Local explanation (Lime explainer, SHAP force plot)

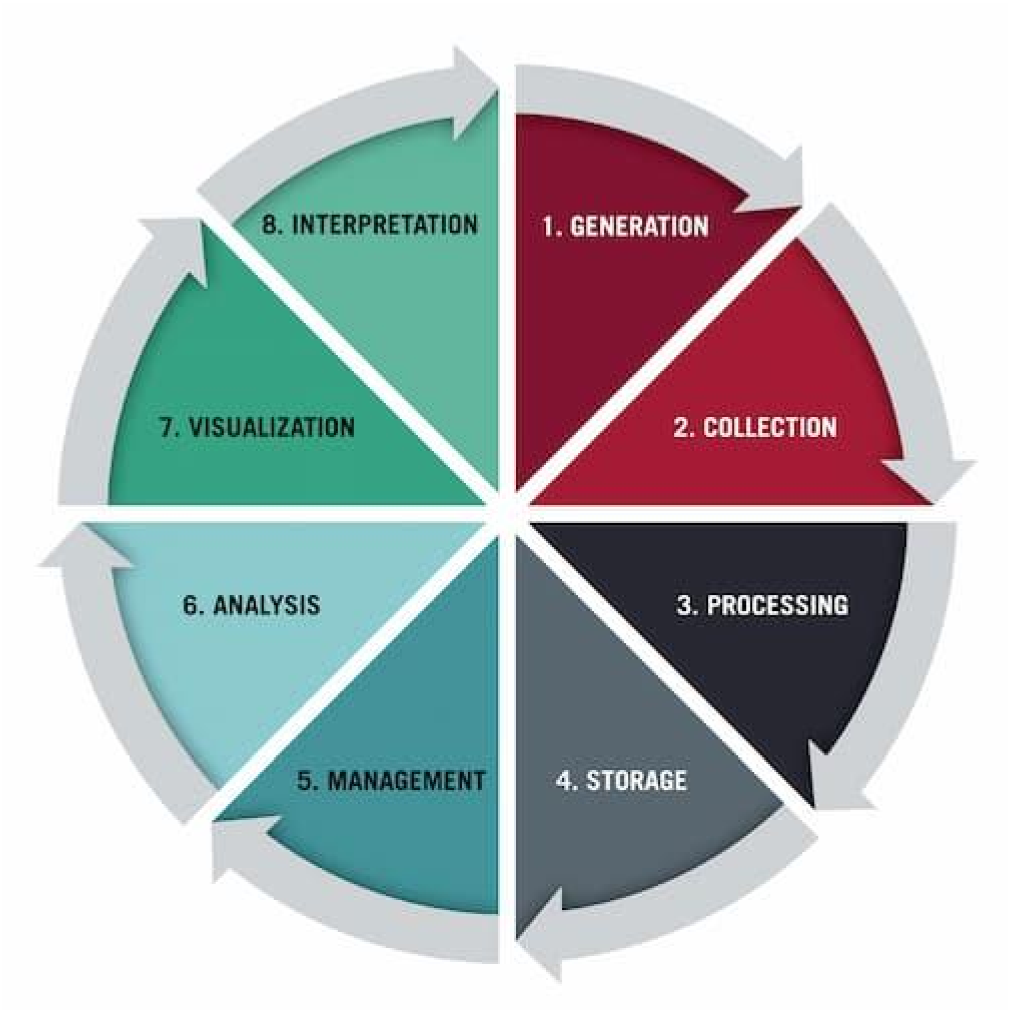
Tools Used:

TensorFlow: TensorFlow applications can be run conveniently on your local machine, cloud, android and iOS devices. As it is built on a deployable scale, it runs on CPU and GPU.
PyTorch: PyTorch is similar to TensorFlow in terms of the nature of the projects chosen. However, when the priority is for faster development, PyTorch is the better choice. TensorFlow is gone in case the project involves larger and more complex project.
Scikit Learn: Scikit-learn is a widely praised Artificial Intelligence tool that simplifies the complexities of machine learning tasks. It boasts an intuitive and use-friendly interface that caters to learners across different proficiency levels. Scikit-learn equips users with the means to construct
and deploy machine learning models effortlessly.
Natural Language Processing: It focuses on developing natural interactions between humans and computers. Specialized software helps machines process human language, create understandable words, and interact with humans through language.
Figstack: Figstack provides a comprehensive set of artificial intelligence tools designed to support developers in comprehending and documenting code more effectively. Its diverse array of features is geared towards

simplifying the coding process, featuring a natural language interpreter
capable of understanding code in nearly any programming language.

Data Collection:

Data collection is the process of collecting and analyzing information on relevant variables in a predetermined, methodical way so that one can respond to specific research questions, test hypotheses, and assess results.

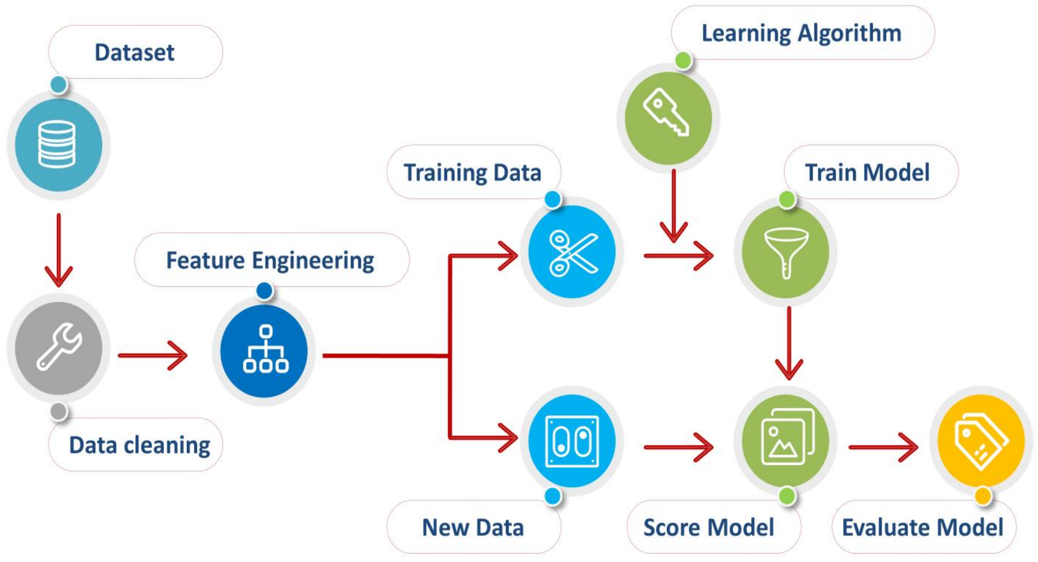


DATA PREPROCESSING :

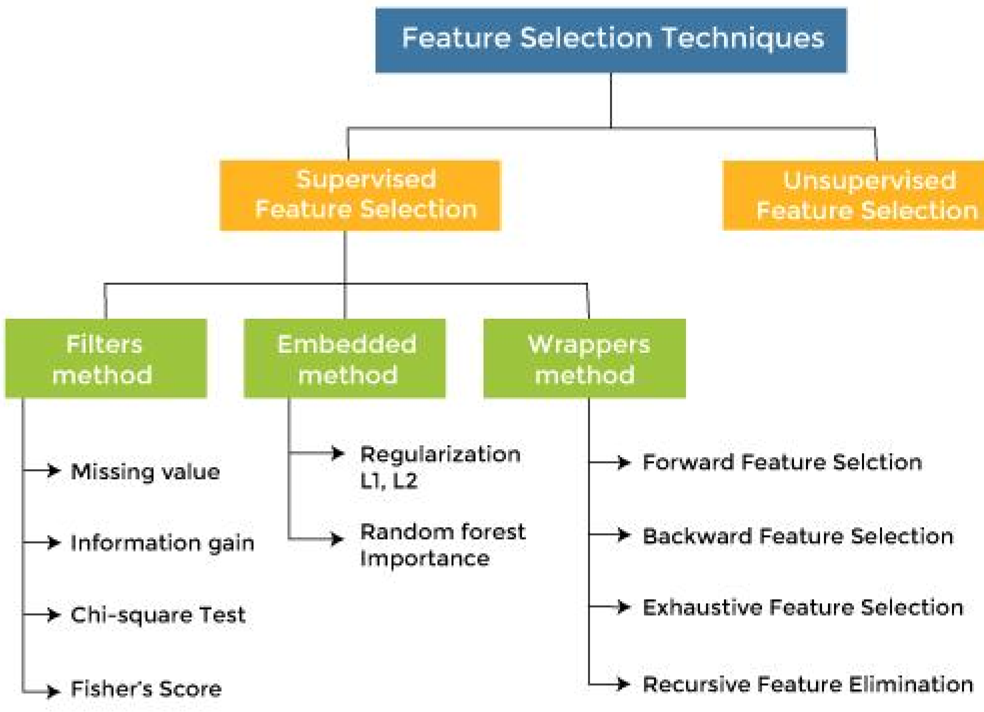
Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis.

The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task.

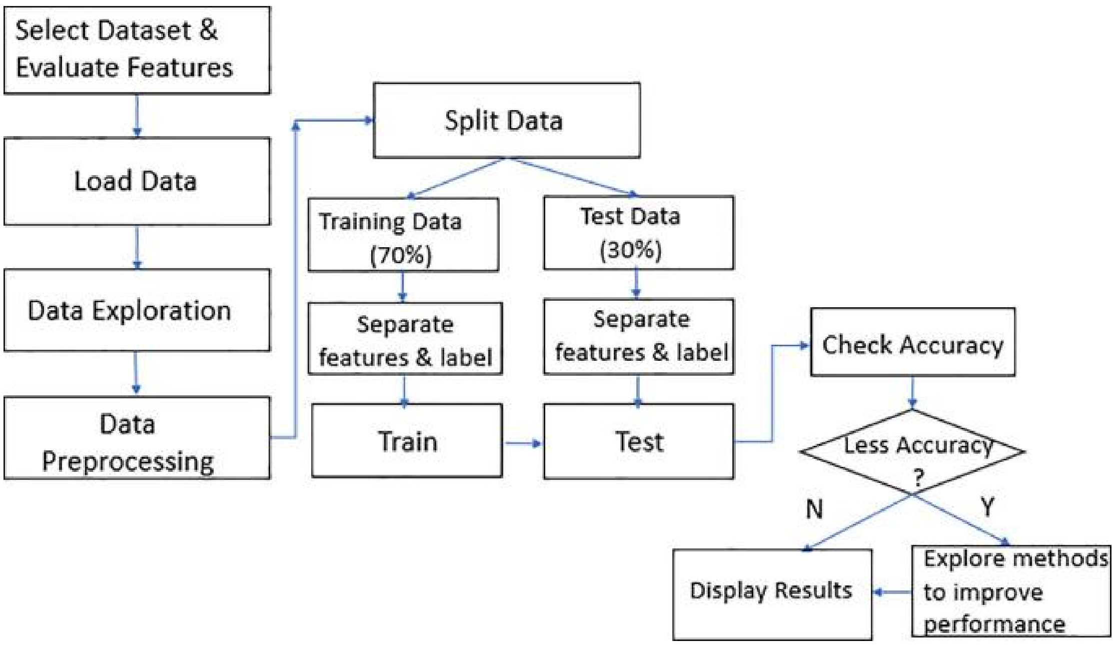
Data preprocessing is an important step in the data mining process that involves cleaning and transforming raw data to make it suitable for analysis.



Feature Selection:



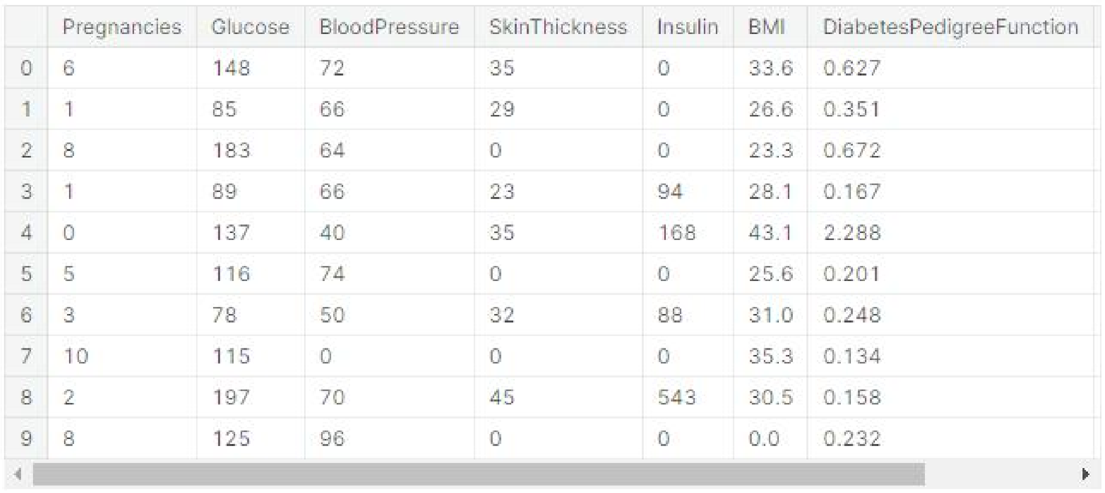
MODEL SELECTION :



CODE :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')
```

OUTPUT:

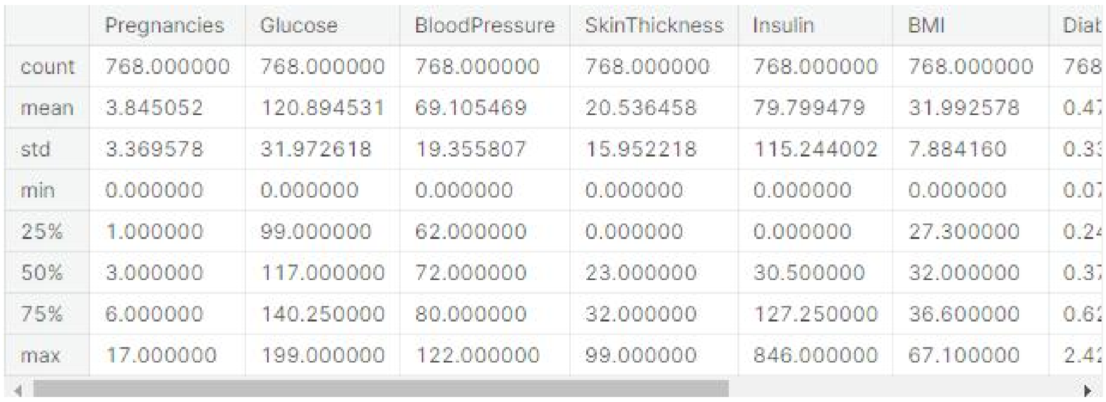


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
5	5	116	74	0	0	25.6	0.201
6	3	78	50	32	88	31.0	0.248
7	10	115	0	0	0	35.3	0.134
8	2	197	70	45	543	30.5	0.158
9	8	125	96	0	0	0.0	0.232

CODE:

```
df.describe()
```

OUTPUT :

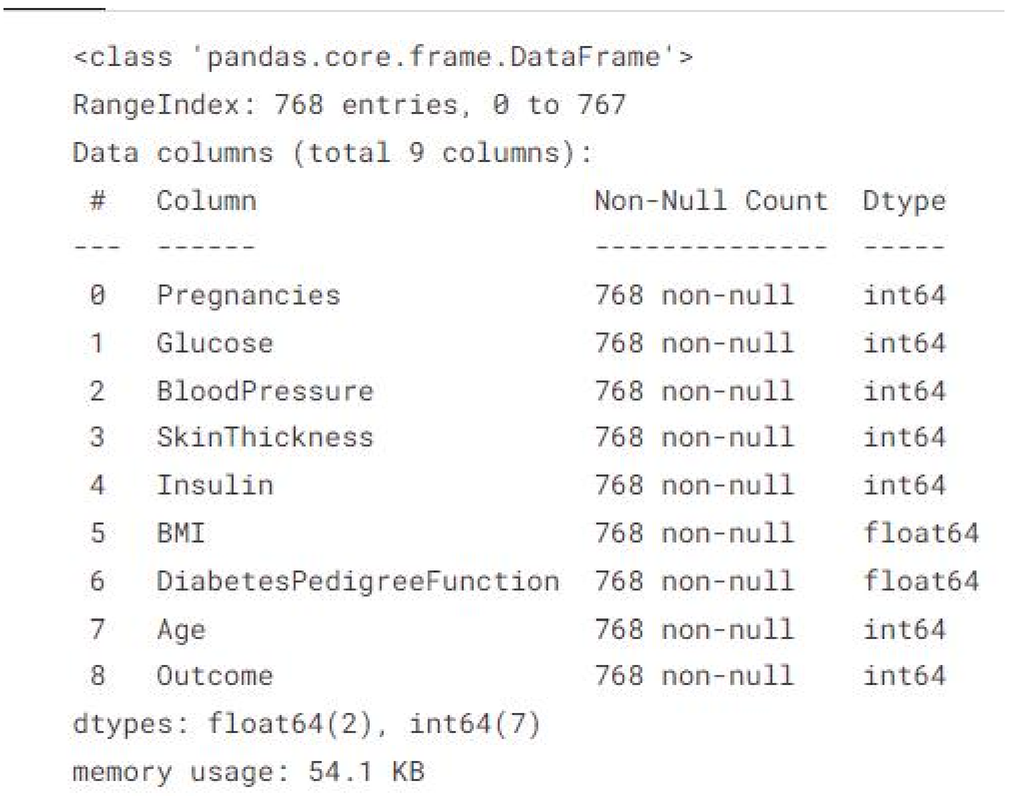


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diat
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.47
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.33
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.07
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.24
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.37
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.62
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.42

CODE :

```
df.info()
```

OUTPUT :



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

CODE :

```
df=df.drop_duplicates()
df. shape()
```

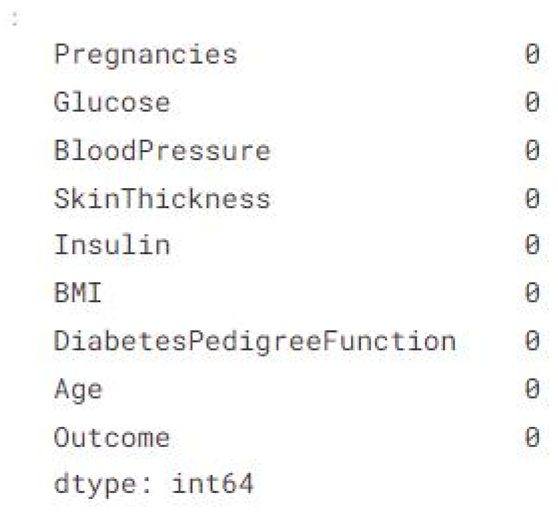
OUTPUT :

```
(768, 9)
```

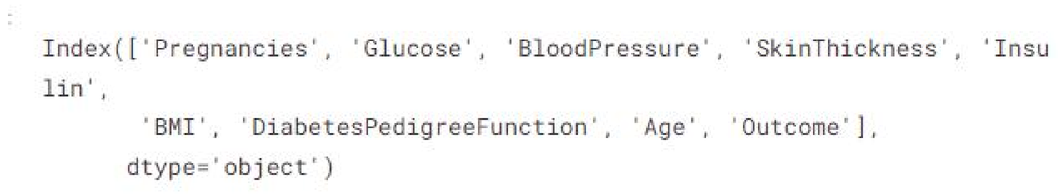
CODE :

```
df.isnull().sum()
df.columns()
```

OUTPUT :



```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```



```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
lin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

<u>DATA VISUALISATION :</u>

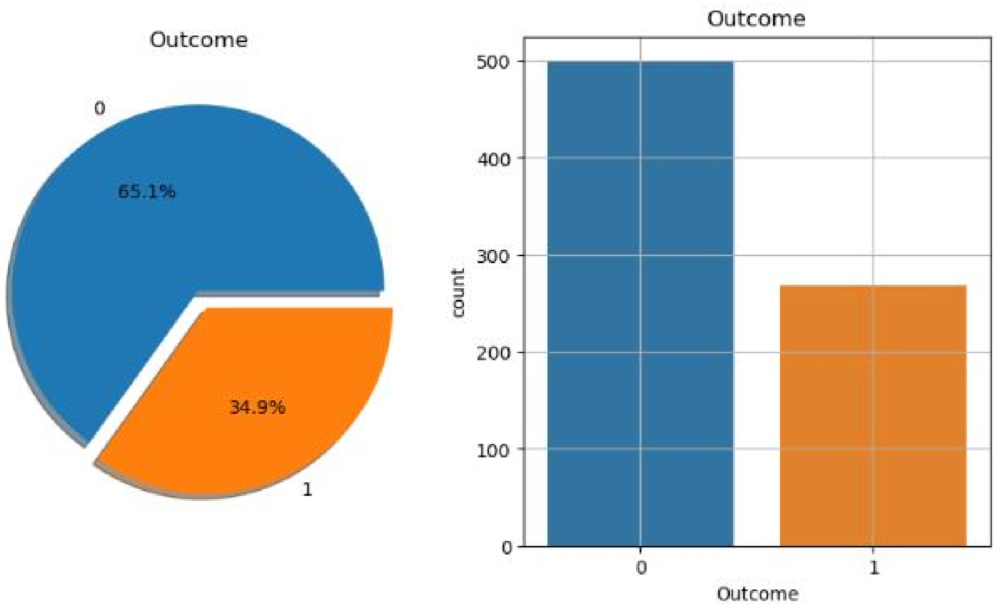
CODE :

```
f, ax = plt.subplots(1, 2, figsize=(10, 5))
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1],
autopct='%1.1f%%', ax=ax[0], shadow=True)
```

```
    ax[0].set_title('Outcome')
    ax[0].set_ylabel(' ')
    sns.countplot(x='Outcome', data=df, ax=ax[1]) # Use 'x'
instead of 'Outcome'
    ax[1].set_title('Outcome')# Display class distribution N, P =
df['Outcome'].value_counts()
    print('Negative (0):', N)
    print('Positive (1):', P)
    plt.grid()
    plt.show()
```

OUTPUT :

```
Negative (0): 500
Positive (1): 268
```

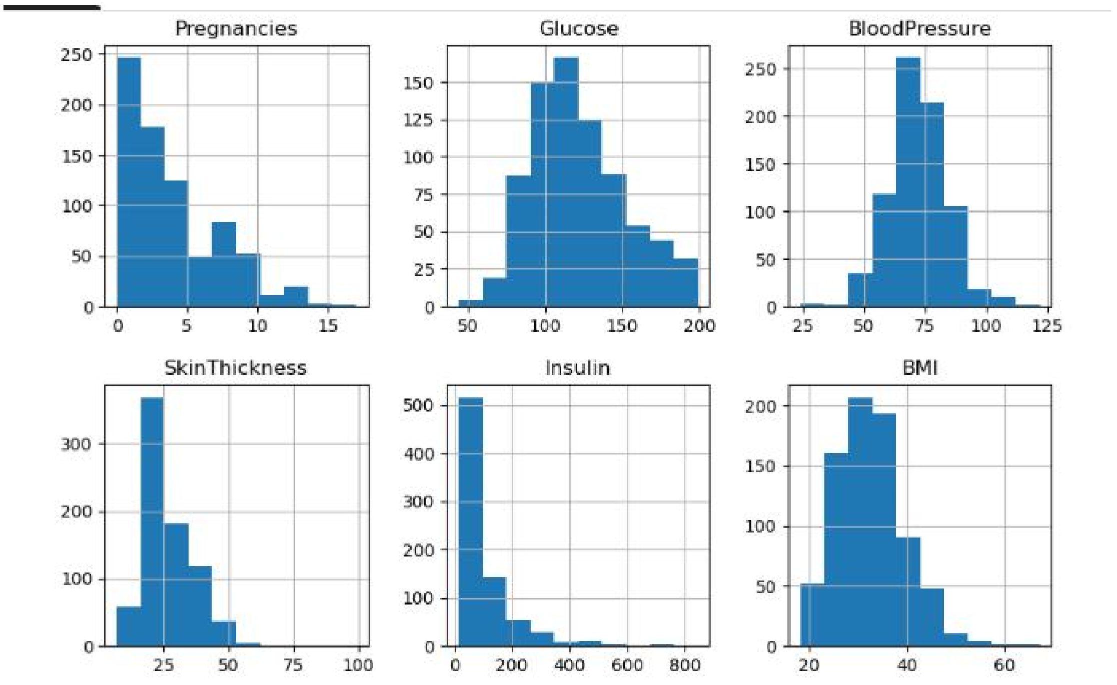


HISTOGRAM :

CODE :

```
  df.hist(bins=10, figsize=(10, 10))
  plt.show()
```

OUTPUT :

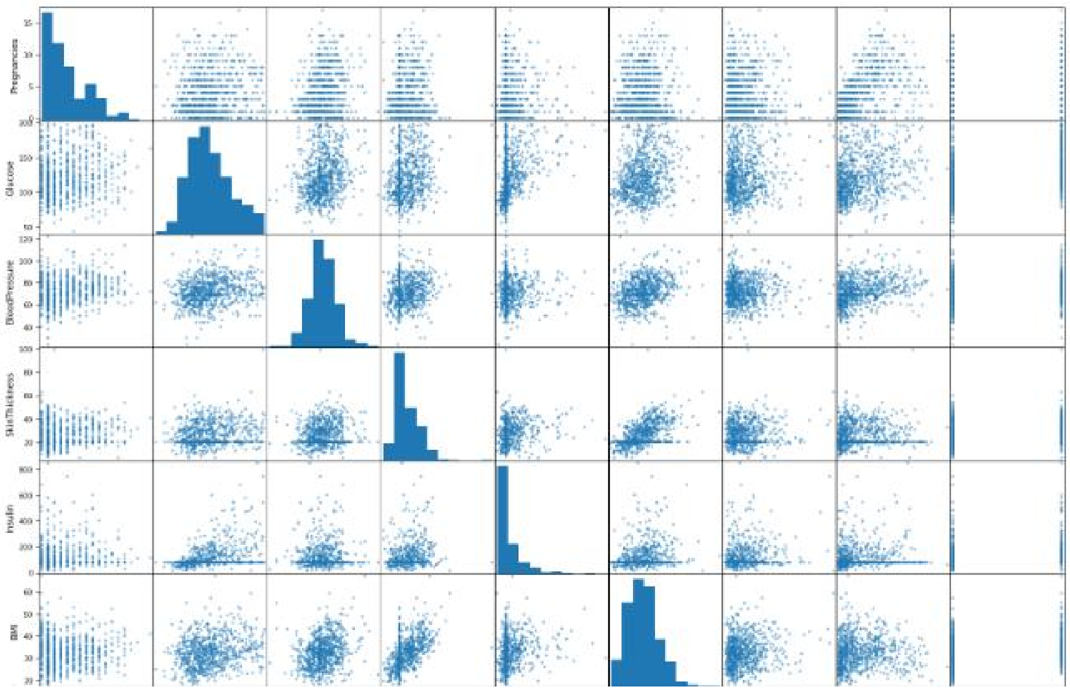


SCATTER PLOT :

CODE :

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize =(20, 20))
```

OUTPUT :



PAIR PLOT :

You can create a pair plot in a Pandas data frame using the pairplot() function from Seaborn.

By customizing the arguments in the pairplot() function, you can customize the appearance of the pair plot to better suit your needs.

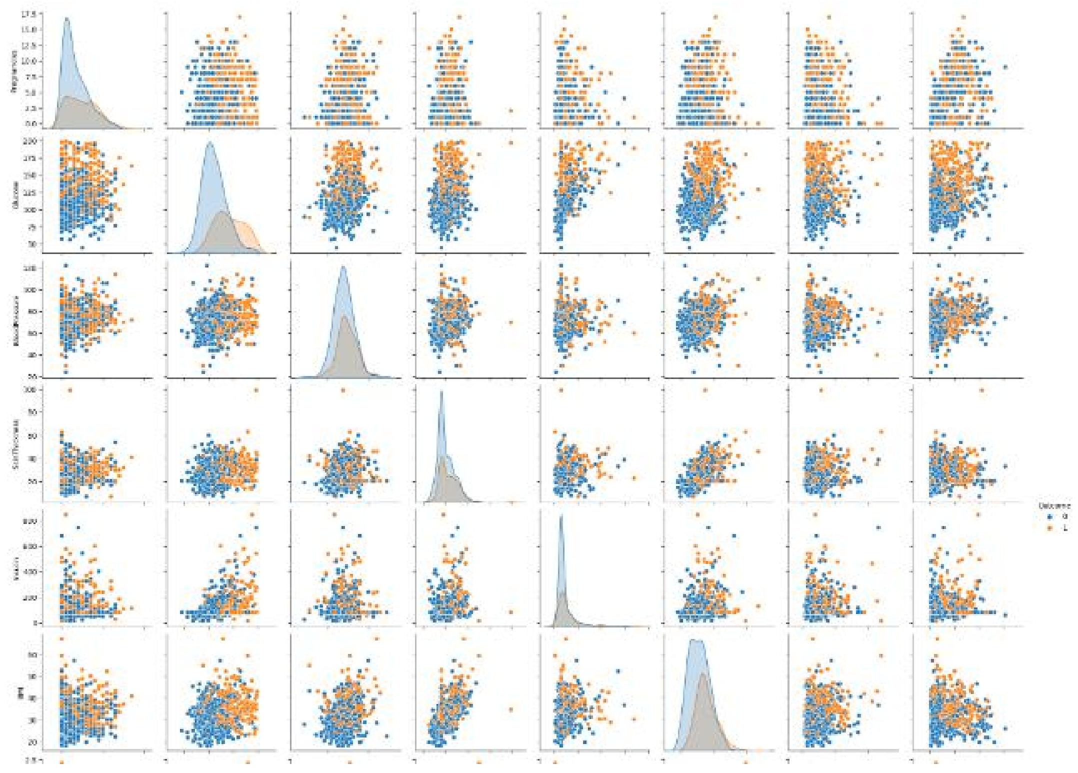
The Seaborn Pairplot allows us to plot pairwise relationships between variables within a dataset.

This creates a nice visualisation and helps us understand the data by summarising a large amount of data in a single figure.

CODE :

```
sns.pairplot(data=df, hue='Outcome')
plt.show()
```

OUTPUT :

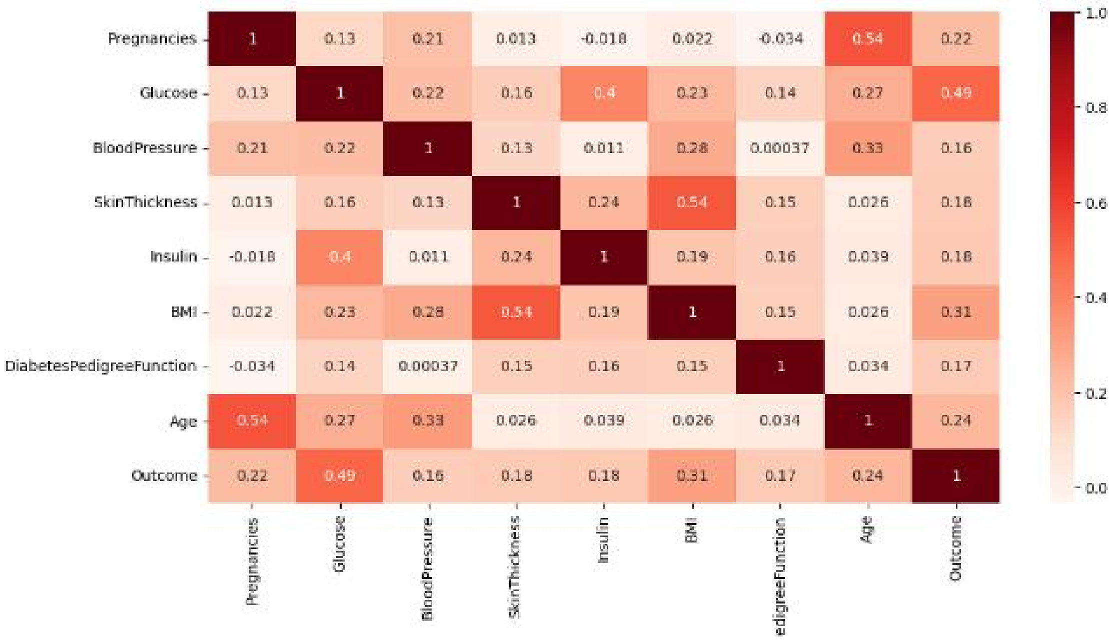


HEATMAP :

CODE :

```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
```

OUTPUT :



Check null Values:

CODE :

```
df.size()
df.isnull().sum()
```

OUTPUT :

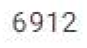
```
6912
```

Check the number of Zero Values in Dataset :

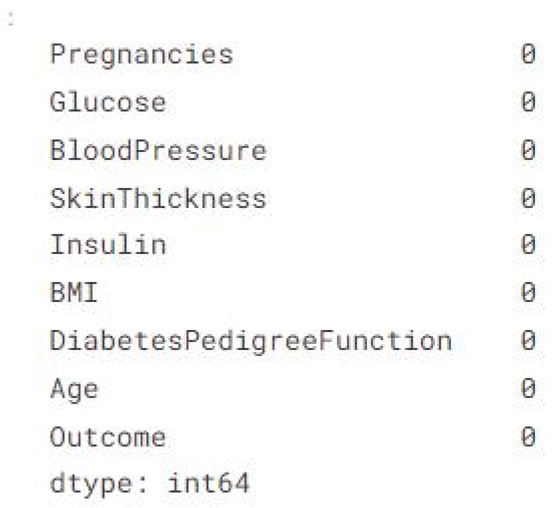
CODE :

```
print("No. of Zero Values in Glucose ",
df[df['Glucose']==0].shape[0])
```

OUTPUT :



```
6912
```



```
:
  Pregnancies                 0
  Glucose                     0
  BloodPressure               0
  SkinThickness               0
  Insulin                     0
  BMI                         0
  DiabetesPedigreeFunction    0
  Age                         0
  Outcome                     0
  dtype: int64
```

CODE :

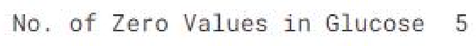
```
print("No. of Zero Values in Blood Pressure
df[df['BloodPressure']==0].shape[0])
```

OUTPUT :

CODE :

```
print("No. of Zero Values in SkinThickness ",
df[df['SkinThickness']==0].shape[0])
```

OUTPUT :

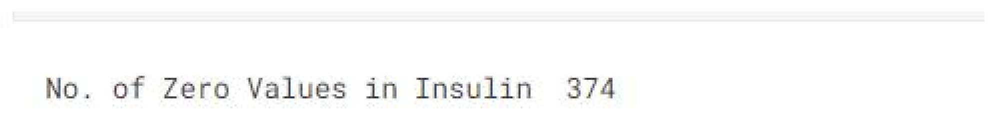


```
No. of Zero Values in Glucose  5
```

CODE :

```
print("No. of Zero Values in Insulin ",
df[df['Insulin']==0].shape[0])
```

OUTPUT :



```
No. of Zero Values in Insulin  374
```

CODE :

```
print("No. of Zero Values in BMI ",
df[df['BMI']==0].shape[0])
```

OUTPUT :

```
No. of Zero Values in BMI  11
```

Replace zeroes with mean of that Columns :

CODE :

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
print('No of zero Values in Glucose ',
df[df['Glucose']==0].shape[0])
```

OUTPUT :

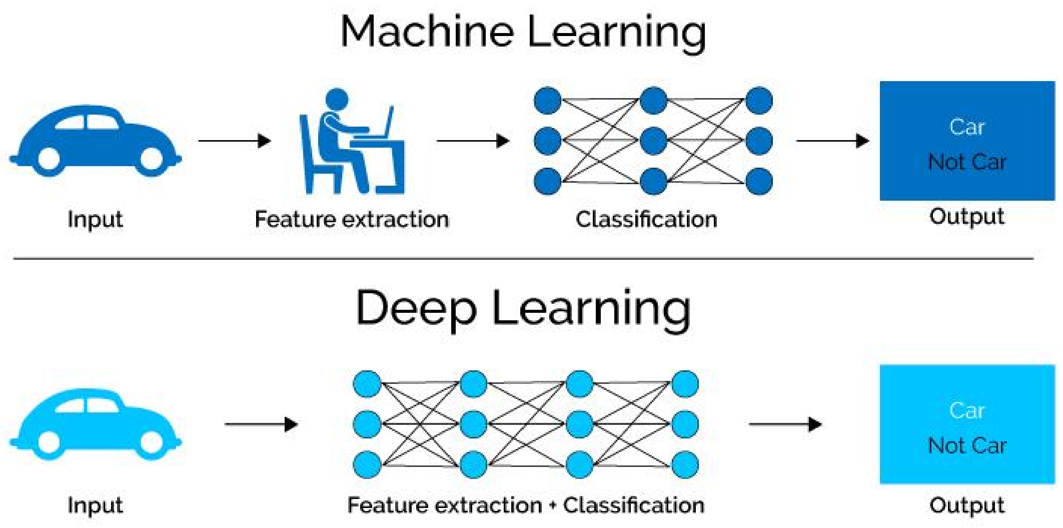
```
No of zero Values in Glucose  0
```

CODE :

```
        df['BloodPressure']=df['BloodPressure'].replace(0,
df['BloodPressure'].mean())
        df['SkinThickness']=df['SkinThickness'].replace(0,
df['SkinThickness'].mean())
        df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
        df['BMI']=df['BMI'].replace(0, df['BMI'].mean())
```

DEEPLEARNING :

Convolutional neural networks (CNNs) : used primarily in computer vision and
image classification applications, can detect features and patterns within an image,
enabling tasks, like object detection or recognition. In 2015, a CNN bested a human in
an object recognition challenge for the first time.
Recurrent neural network (*RNNs*): *are typically used in natural language and*
speech recognition applications as it leverages sequential or times series data.
Deep learning is a subset of machine learning, which is essentially a neural
network with three or more layers. These neural networks attempt to simulate the
behavior of the human brain—albeit far from matching its ability—allowing it to
"learn" from large amounts of data.
Because deep learning models process information in ways similar to the human
brain, they can be applied to many tasks people do. Deep learning is currently used in
most common image recognition tools, natural language processing (NLP) and speech
recognition software.



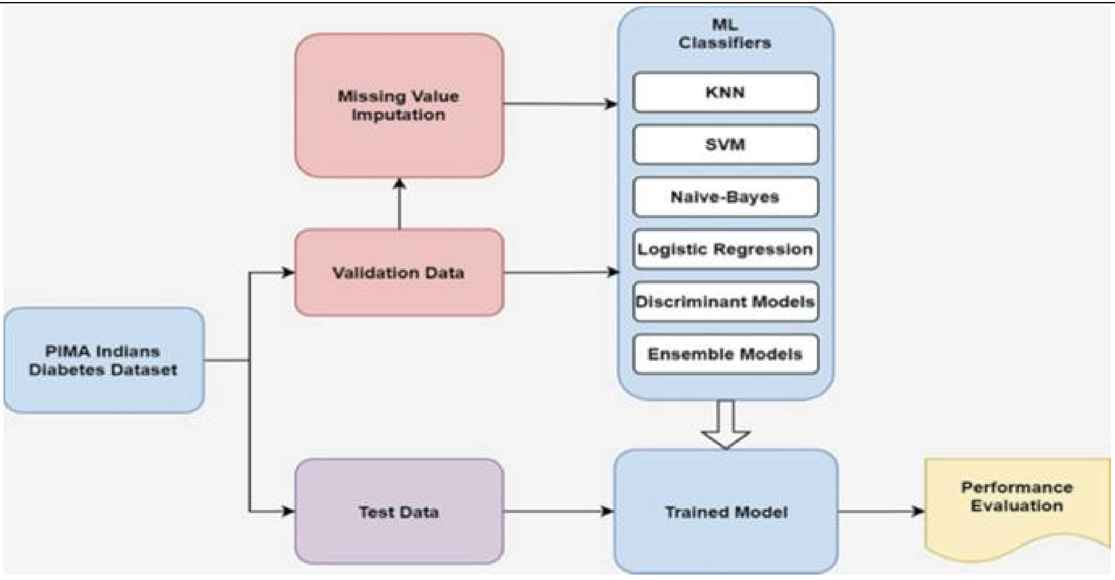
ENSEMBLE METHOD :

The ensemble approach has been proposed using a soft voting classifier to classify diabetes.

The logistic regression Naïve Bayes and Random Forest algorithms have been ensembled and the performance of the ensemble approach has achieved better results as compared to base classifiers.

Ensemble learning methods work by combining the mapping functions learned by contributing members. Ensembles for classification are best understood by the combination of decision boundaries of members.

Ensembles for regression are best understood by the combination of hyperplanes of members.



RANDOM FOREST ALGORITHM :

Random forest is a supervised machine learning algorithm that's used for classification and regression problems.

It's based on the concept of bagging, which involves training a group of models on different subsets of a dataset.

The final output is generated by combining the outputs of all the different models.

The random forest algorithm works by:

- Building decision trees from various samples
- Using the majority vote for classification and average for regression
- Selecting data for each tree using a method called bagging
- Training multiple decision trees in parallel
- Determining the final output via a majority vote

The Random Forest Algorithm is commonly used because it’s easy to use
and flexible .It can be used for classification and regression problems, such
as:

- Classifying whether an email is “spam” or “not spam”
- Handling both classification and regression problems.

WORKING OF RANDOM FOREST ALGORITHM :

The following steps explain the working Random Forest Algorithm:

Step 1 : Select random samples from a given data or training set.
Step 2 : This algorithm will construct a decision tree for every training data.
Step 3 : Voting will take place by averaging the decision tree.
Step 4 : Finally, select the most voted prediction result as the final prediction result. This combination of multiple models is called Ensemble. Ensemble uses two methods:

1. **Bagging:** Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.
2. **Boosting:** Combing weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.

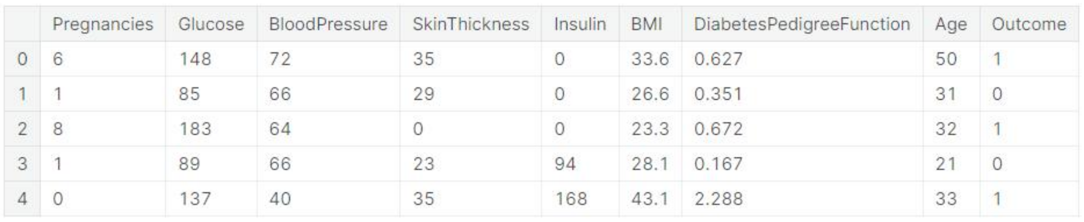
Bagging: From the principle mentioned above, we can understand Random forest uses the Bagging code. Now, let us understand this concept in detail. Bagging is also known as Bootstrap Aggregation used by random forest. The process beginswith any original random data. After arranging, it is organised into samples known as Bootstrap Sample. This process is known as Bootstrapping.Further, the models are trained individually, yielding different results known as Aggregation. In the last step, all the results are combined, and the generated output is based on
majority voting. This step is known as Bagging and is done using an Ensemble Classifier.

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score,confusion_matrix,
precision_score, recall_score
# Load the dataset
    data = pd.read_csv("/kaggle/input/diabetes-data-
set/diabetes.csv")
    data.head()
```

OUTPUT:

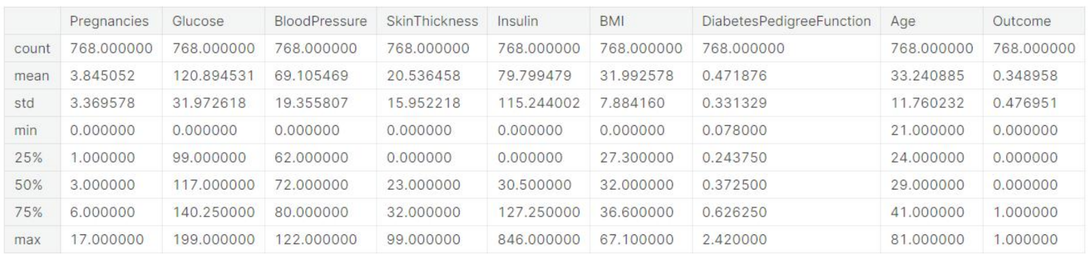


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

CODE:

```
        # Perform exploratory data analysis
        # Summary statistics
        summary_stats = data.describe()
        summary_stats
```

OUTPUT:



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

CODE:

```
        # Class distribution
        class_distribution = data['Outcome'].value_counts()
        class_distribution
```

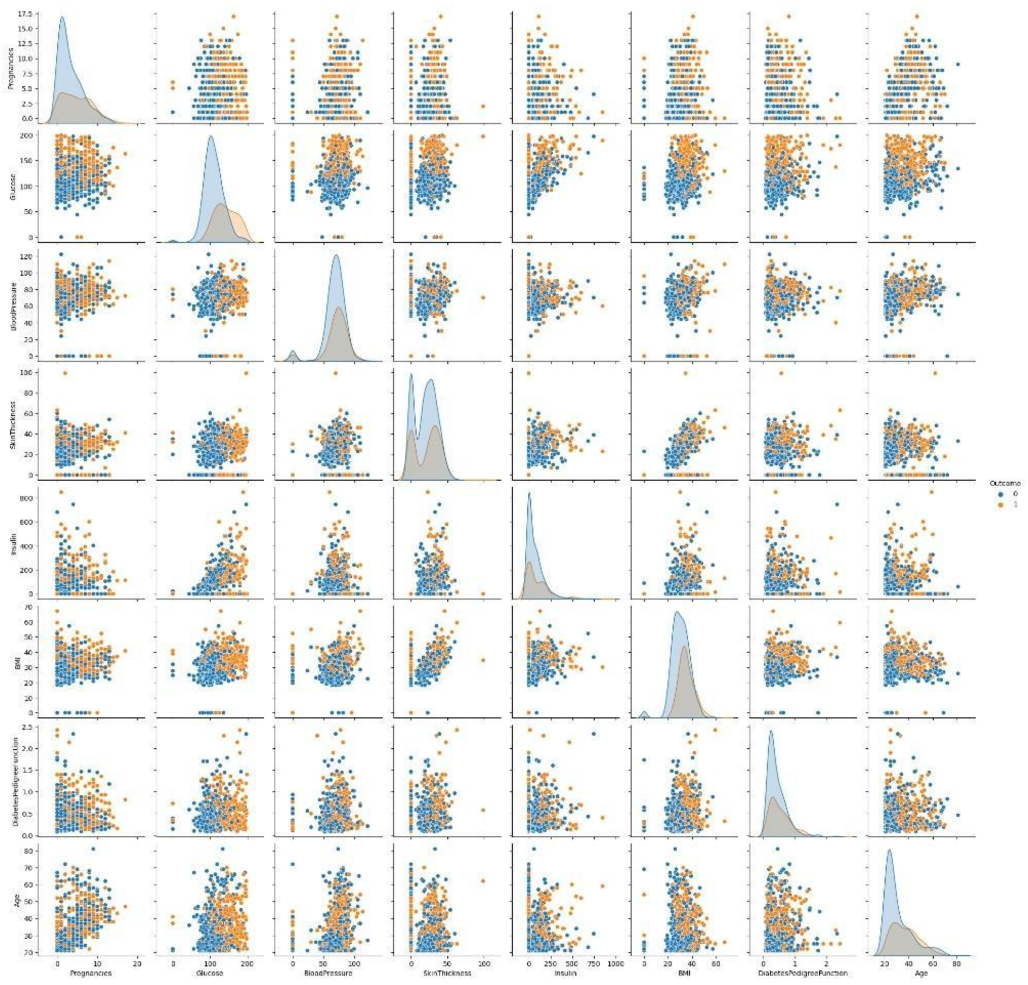
OUTPUT:

```
Outcome
0
500
1
268
Name: count, dtype: int64
```

CODE:

```
# Pairplot for visualizing relationships between features
sns.pairplot(data, hue='Outcome', diag_kind='kde')
plt.show()
```

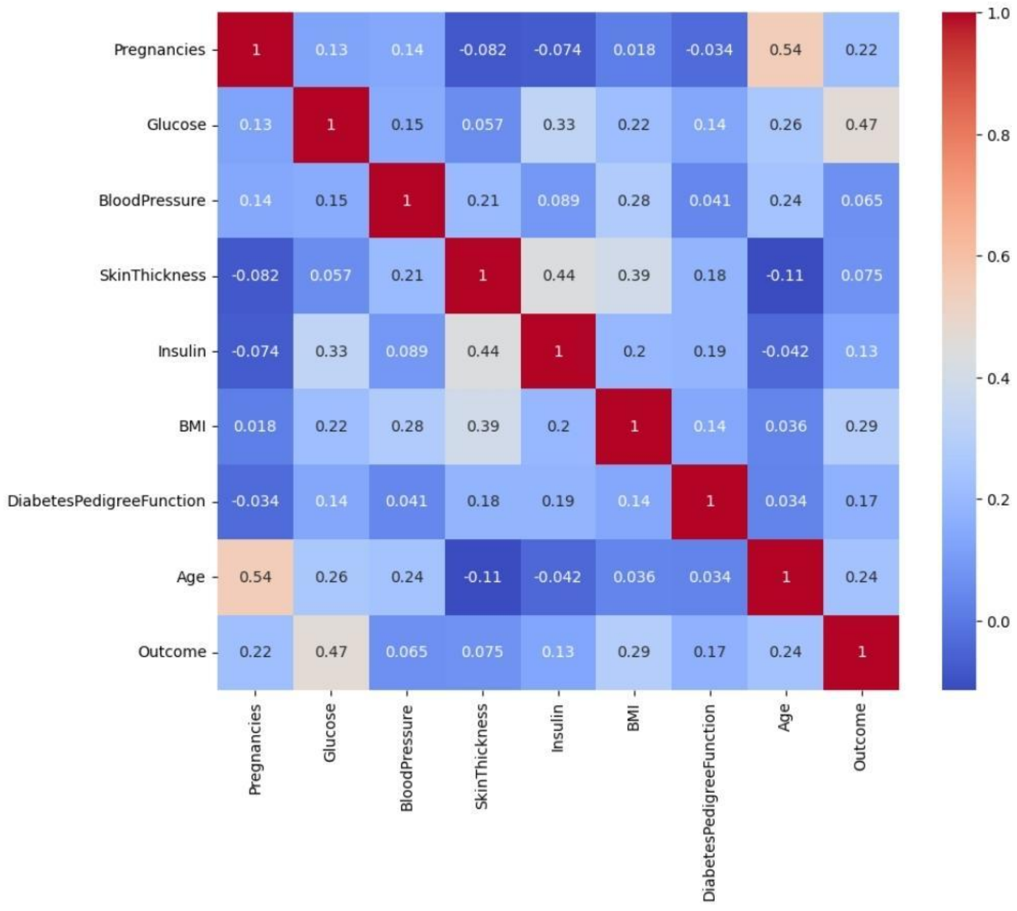
OUTPUT:



CODE:

```
        # Correlation heatmap
        correlation_matrix = data.corr()
        plt.figure(figsize=(10, 8))
        sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm')
        plt.show()
```

OUTPUT:



FUTURE SCALING :

CODE :

```
# Standard Scaler:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)from sklearn.model_selection
import train_test_split
```

```
 X_train, X_test, y_train, y_test = train_test_split(SSX, y,
test_size=0.2, random_state=7)
```

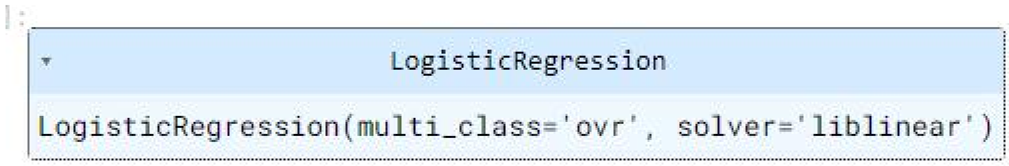
Classification Algorithms:

Logistic Regression:

CODE :

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

OUTPUT :



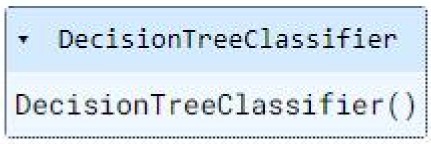
```
LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

Descision Tree:

CODE :

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

OUTPUT :



```
DecisionTreeClassifier
DecisionTreeClassifier()
```

CODE:

```
# Make predictions on the testing data
y_pred = rf_classifier.predict(X_test)
# Evaluate the model's performance
```

```
        accuracy = accuracy_score(y_test, y_pred)
        confusion = confusion_matrix(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        # Calculate specificity
        tn, fp, fn, tp = confusion.ravel()
        specificity = tn / (tn + fp)
        # Print the results
        print("Accuracy:", accuracy)
        print("Confusion Matrix:")
        print(confusion)
        print("Precision:", precision)
        print("Recall:", recall)
        print("Specificity:", specificity)
```

OUTPUT:

```
        Accuracy: 0.7532467532467533
        Confusion Matrix:
        [[121 30]
        [ 27 53]]
        Precision: 0.6385542168674698
        Recall: 0.6625
        Specificity: 0.8013245033112583
```

<u>Split the DataFrame into X and y:</u>

CODE:

```
target_name='Outcome'

 y=df[target_name]

 X= df.drop(target_name, axis=1)
```

```
X.head()
```

OUTPUT:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

CODE:

```
y.head()
```

OUTPUT:

```
0    1
1    0
2    1
3    0
```

```
4    1
Name: Outcome, dtype: int64
```

Making prediction:

Logistic Regression:

CODE :

```
X_test.shape
```

OUTPUT:

```
(154, 8)
```

CODE:

```
lr_pred=lr.predict(X_test)
lr_pred.shape
```

OUTPUT:

```
(154,)
```

Decision Tree:

CODE:

```
dt_pred=dt.predict(X_test)
dt_pred.shape
```

OUTPUT:

```
(154,)
```

Model Evaluation for Logistic Regression:

CODE:

Train Score and Test Score

In [47]:

```
# For Logistic Regression:

from sklearn.metrics import accuracy_scoreprint("Train Accuracy of Logistic Regression: ", lr.score(X_train, y_train)*100)print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test, y_test)*100)print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test, lr_pred)*100)
```

OUTPUT:

```
Train Accuracy of Logistic Regression:  77.36156351791531

Accuracy (Test) Score of Logistic Regression:  77.27272727272727

Accuracy Score of Logistic Regression:  77.27272727272727
```

CODE:

```
# For Decesion Tree:

print("Train Accuracy of Decesion Tree: ", dt.score(X_train, y_train)*100)print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test, y_test)*100)print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test, dt_pred)*100)
```

OUTPUT:

```
Train Accuracy of Decesion Tree:  100.0

Accuracy (Test) Score of Decesion Tree:  80.51948051948052

Accuracy Score of Decesion Tree:  80.51948051948052
```

Confusion Matrix

- *Confusion Matrix of "Logistic Regression"*

CODE:

```
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, lr_pred)cm
```

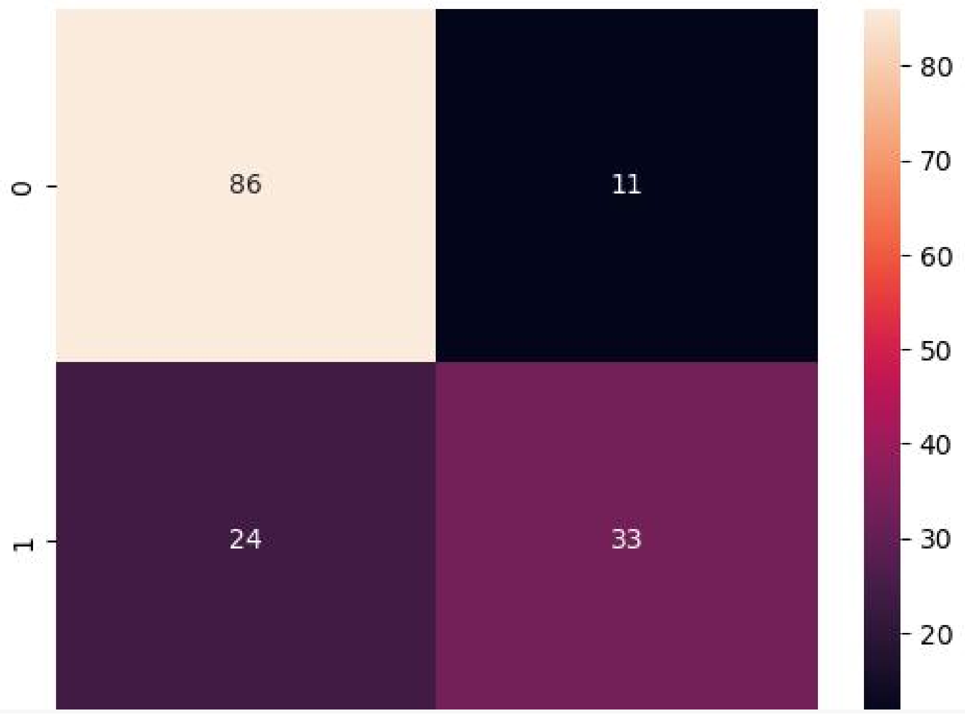
OUTPUT:

```
array([[86, 11],
       [24, 33]])
```

CODE:

```
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")
```

```
<Axes: >
```



CODE:

```
TN =cm[0, 0]FP =cm[0,1]FN = cm[1,0]TP  = cm[1,1]
```

```
TN, FP, FN, TP
```

OUTPUT:

```
(86, 11, 24, 33)
```

CODE:

```
from sklearn.metrics import classification_report, confusion_matrixfrom sklearn.metrics import accuracy_score, roc_auc_score, roc_curvecm = confusion_matrix(y_test, lr_pred)
```

```
print('TN - True Negative {}'.format(cm[0,0]))print('FP - False Positive {}'.format(cm[0,1]))print('FN - False Negative {}'.format(cm[1,0]))print('TP - True Positive {}'.format(cm[1,1]))print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.sum(cm))*100))print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]), np.sum(cm))*100))
```

OUTPUT:

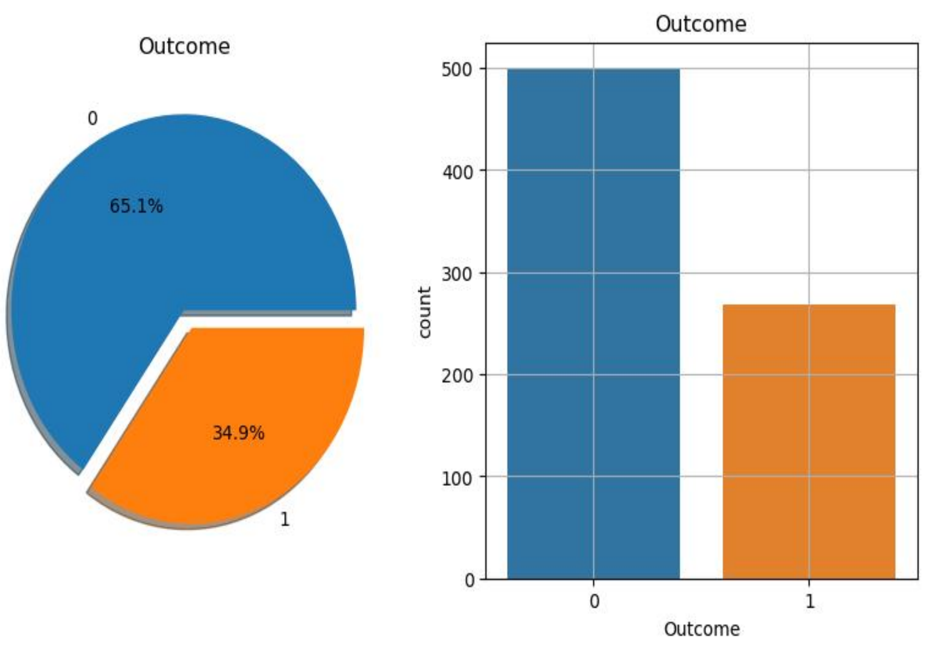
```
TN - True Negative 86

FP - False Positive 11

FN - False Negative 24

TP - True Positive 33

Accuracy Rate: 77.27272727272727

Misclassification Rate: 22.727272727272727
```

```
77.27272727272727+22.727272727272727
```

```
100.0
```

CODE:

```
import matplotlib.pyplot as plt

import numpy as np

plt.clf()

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)

classNames = ['0', '1']plt.title('Confusion Matrix of Logistic Re
gression')

plt.ylabel('Actual (true) Values')

plt.xlabel('Predicted Values')

tick_marks = np.arange(len(classNames))

plt.xticks(tick_marks, classNames, rotation=45)

plt.yticks(tick_marks, classNames)

s = [['TN', 'FP'], ['FN', 'TP']]for i in range(2):

    for j in range(2):

        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```



CODE:

```
pd.crosstab(y_test, lr_pred, margins=False)
```

OUTPUT:

col_0	0	1
Outcome		
0	86	11
1	24	33

CODE:

```
pd.crosstab(y_test, lr_pred, margins=True)
```

OUTPUT:

col_0	0	1	All
Outcome			
0	86	11	97
1	24	33	57

col_0	0	1	All
Outcome			
All	110	44	154

CODE:

```
pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted values'], margins=True)
```

OUTPUT:

0	1	All
86	11	97
24	33	57
110	44	154

Precision:

PPV- positive Predictive Value

Precision = True Positive/True Positive + False Positive
Precision = TP/TP+FP

CODE:

```
TP, FP
```

OUTPUT:

```
(33, 11)
```

CODE:

```
Precision = TP/(TP+FP)Precision
```

OUTPUT:

```
0.75
```

In [61]:

```
33/(33+11)
```

Out[61]:

```
0.75
```

In [62]:

```
# precision Score:

precision_score = TP/float(TP+FP)*100print('Precision Score: {0:0.4f}'.format(precision_score))
```

```
Precision Score: 75.0000
```

In [63]:

```
from sklearn.metrics import precision_score

print("Precision Score is: ", precision_score(y_test, lr_pred)*100)

print("Micro Average Precision Score is: ", precision_score(y_test, lr_pred, average='micro')*100)

print("Macro Average Precision Score is: ", precision_score(y_test, lr_pred, average='macro')*100)

print("Weighted Average Precision Score is: ", precision_score(y_test, lr_pred, average='weighted')*100)

print("precision Score on Non Weighted score is: ", precision_score(y_test, lr_pred, average=None)*100)
```

```
Precision Score is:  75.0
```

```
Micro Average Precision Score is:  77.27272727272727

Macro Average Precision Score is:  76.5909090909091

Weighted Average Precision Score is:  77.00413223140497

precision Score on Non Weighted score is:  [78.18181818 75.
  ]
```

CODE:

```
print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

Classification Report of Logistic Regression:

```
              precision    recall  f1-score   support


           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57


    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

Recall

True Positive Rate(TPR)

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN

In [65]:

```
recall_score = TP/ float(TP+FN)*100print('recall_score', recall
_score)
```

```
recall_score 57.89473684210527
```

In [66]:

```
TP, FN
```

Out[66]:

```
(33, 24)
```

In [67]:

```
33/(33+24)
```

Out[67]:

```
0.5789473684210527
```

In [68]:

```
from sklearn.metrics import recall_scoreprint('Recall or Sensiti
vity_Score: ', recall_score(y_test, lr_pred)*100)
```

```
Recall or Sensitivity_Score:  57.89473684210527
```

```
print("recall Score is: ", recall_score(y_test, lr_pred)*100)

print("Micro Average recall Score is: ", recall_score(y_test, lr_
pred, average='micro')*100)

print("Macro Average recall Score is: ", recall_score(y_test, lr_
pred, average='macro')*100)

print("Weighted Average recall Score is: ", recall_score(y_test,
 lr_pred, average='weighted')*100)
```

```
print("recall Score on Non Weighted score is: ", recall_score(y
_test, lr_pred, average=None)*100)
```

```
recall Score is:  57.89473684210527

Micro Average recall Score is:  77.27272727272727

Macro Average recall Score is:  73.27726532826912

Weighted Average recall Score is:  77.27272727272727

recall Score on Non Weighted score is:  [88.65979381 57.89473
684]
```

In [70]:

```
print('Classification Report of Logistic Regression: \n', classifi
cation_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support



           0     0.7818    0.8866    0.8309        97

           1     0.7500    0.5789    0.6535        57



    accuracy                         0.7727       154

   macro avg     0.7659    0.7328    0.7422       154

weighted avg     0.7700    0.7727    0.7652       154
```

FPR - False Positve Rate

In [71]:

```
FPR = FP / float(FP + TN) * 100print('False Positive Rate: {:.4f}'.format(FPR))
```

```
False Positive Rate: 11.3402
```

In [72]:

```
FP, TN
```

Out[72]:

```
(11, 86)
```

In [73]:

```
11/(11+86)
```

Out[73]:

```
0.1134020618556701
```

Specificity:

In [74]:

```
specificity = TN /(TN+FP)*100print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 88.6598
```

In [75]:

```
from sklearn.metrics import f1_scoreprint('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

```
F1_Score of Macro:  65.34653465346535
```

In [76]:

```
print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred, average='micro')*100)print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred, average='macro')*100)print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred, average='weighted')*100)print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred, average=None)*100)
```

```
Micro Average f1 Score is:  77.27272727272727

Macro Average f1 Score is:  74.21916104653944

Weighted Average f1 Score is:  76.52373933045479

f1 Score on Non Weighted score is:  [83.09178744 65.34653465]
```

Classification Report of Logistic Regression:

CODE:

```
from sklearn.metrics import classification_report

print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support


           0     0.7818    0.8866    0.8309        97

           1     0.7500    0.5789    0.6535        57


    accuracy                         0.7727       154

   macro avg     0.7659    0.7328    0.7422       154

weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve& ROC AUC

***CODE*:**

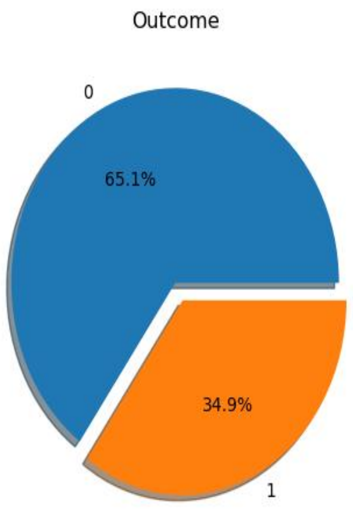
```
# Area under Curve:auc= roc_auc_score(y_test, lr_pred)print("
ROC AUC SCORE of logistic Regression is ", auc)
```

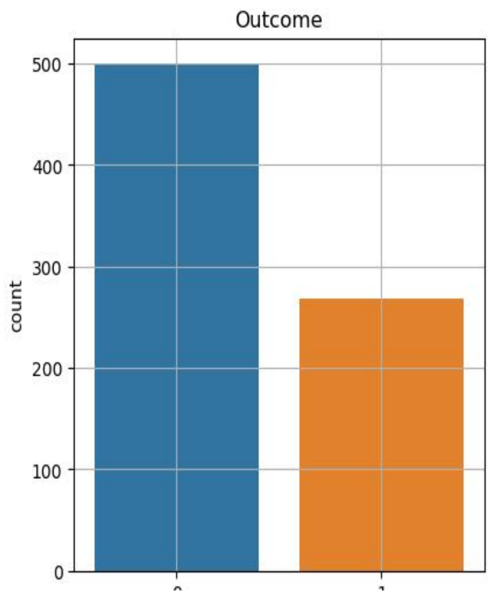
```
ROC AUC SCORE of logistic Regression is  0.7327726532826
913
```

CODE:

```
from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt fpr, tpr, thresholds = roc_curve
(y_test, lr_pred)

plt.plot(fpr, tpr, color='orange', label="ROC")

plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='RO
C curve (area = %0.2f)' % auc(fpr, tpr))

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver Operating Characteristics (ROC) Curve of L
ogistic Regression")plt.legend()plt.grid()plt.show()
```

OUTPUT:





Confusion Matrix:

- Confusion matrix of "Decision Tree"

CODE:

```
from sklearn.metrics import classification_report, confusion_m
atrix
```

```
cm = confusion_matrix(y_test, dt_pred)cm
```

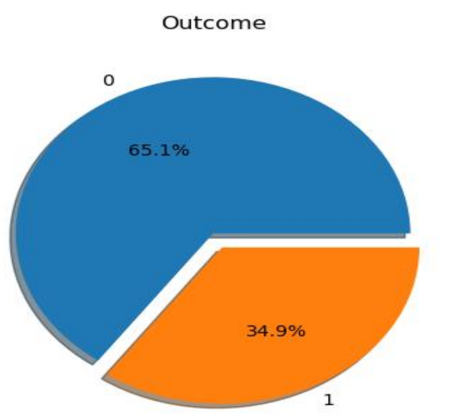
OUTPUT:

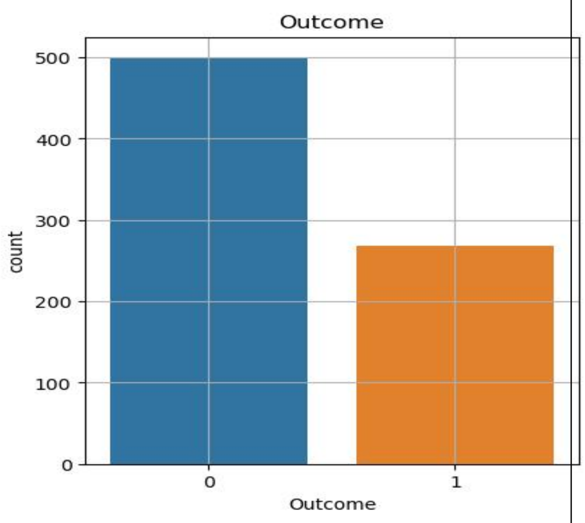
```
array([[81, 16],

       [14, 43]])
```

CODE:

```
sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, f
mt="d")
```

OUTPUT:





CODE:

```
TN =cm[0, 0]FP =cm[0,1]FN = cm[1,0]TP  = cm[1,1]
```

```
TN, FP, FN, TP
```

OUTPUT:

```
(81, 16, 14, 43)
```

CODE:

```
from sklearn.metrics import classification_report, confusion_matrix

from sklearn.metrics import accuracy_score, roc_auc_score, roc_curvecm = confusion_matrix(y_test, dt_pred)

print('TN - True Negative {}'.format(cm[0,0]))

print('FP - False Positive {}'.format(cm[0,1]))

print('FN - False Negative {}'.format(cm[1,0]))

print('TP - True Positive {}'.format(cm[1,1]))

print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]), np.sum(cm))*100))

print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1], cm[1,0]]), np.sum(cm))*100))
```

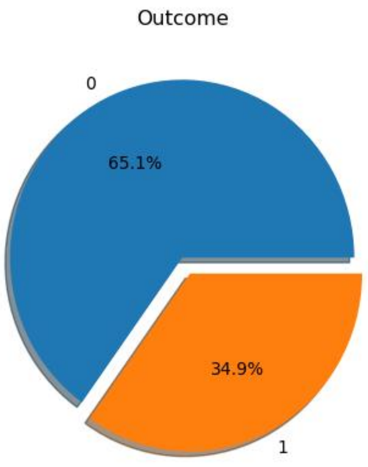
OUTPUT:

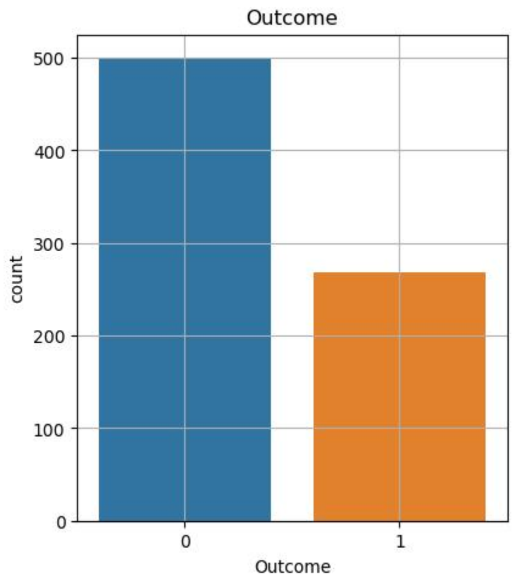
```
TN - True Negative 81

FP - False Positive 16

FN - False Negative 14

TP - True Positive 43

Accuracy Rate: 80.51948051948052
```

```
Misclassification Rate: 19.480519480519483
```

CODE:

```
import matplotlib.pyplot as pltimport numpy as np

plt.clf()plt.imshow(cm, interpolation='nearest', cmap=plt.cm.W
istia)

classNames = ['0', '1']plt.title('Confusion Matrix of Decision Tr
ee')

plt.ylabel('Actual (true) Values')

plt.xlabel('Predicted Values')

tick_marks = np.arange(len(classNames))

plt.xticks(tick_marks, classNames, rotation=45)

plt.yticks(tick_marks, classNames

)s = [['TN', 'FP'], ['FN', 'TP']]for i in range(2):

    for j in range(2):

        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))plt.show()
```





Precision:

CODE:

```
# precision Score:

precision_score = TP/float(TP+FP)*100print('Precision Score: {0:0.4f}'.format(precision_score))
```

OUTPUT:

```
Precision Score: 72.8814
```

CODE:

```
from sklearn.metrics import precision_score

print("Precision Score is:", precision_score(y_test, dt_pred) * 100)

print("Micro Average Precision Score is:", precision_score(y_test, dt_pred, average='micro') * 100)

print("Macro Average Precision Score is:", precision_score(y_test, dt_pred, average='macro') * 100)

print("Weighted Average Precision Score is:", precision_score(y_test, dt_pred, average='weighted') * 100)

print("Precision Score on Non Weighted score is:", precision_score(y_test, dt_pred, average=None) * 100)
```

OUTPUT:

```
Precision Score is: 72.88135593220339

Micro Average Precision Score is: 80.51948051948052
```

```
Macro Average Precision Score is: 79.07225691347011

Weighted Average Precision Score is: 80.68028314237056

Precision Score on Non Weighted score is: [85.26315789 72.88
135593]
```

Recall:

CODE:

```
recall_score = TP/ float(TP+FN)*100

print('recall_score', recall_score)
```

OUTPUT:

```
recall_score 75.43859649122807
```

CODE:

```
from sklearn.metrics import recall_score

print('Recall or Sensitivity_Score: ', recall_score(y_test, dt_pre
d)*100)
```

OUTPUT:

```
   Recall or Sensitivity_Score:  75.43859649122807
```

CODE:

```
print("recall Score is: ", recall_score(y_test, dt_pred)*100

)print("Micro Average recall Score is: ", recall_score(y_test, dt
_pred, average='micro')*100)

print("Macro Average recall Score is: ", recall_score(y_test, dt
_pred, average='macro')*100)
```

```
print("Weighted Average recall Score is: ", recall_score(y_test,
 dt_pred, average='weighted')*100)

print("recall Score on Non Weighted score is: ", recall_score(y
_test, dt_pred, average=None)*100)
```

OUTPUT:

```
recall Score is:  75.43859649122807

Micro Average recall Score is:  80.51948051948052

Macro Average recall Score is:  79.47187556520167

Weighted Average recall Score is:  80.51948051948052

recall Score on Non Weighted score is:  [83.50515464 75.43859
649]
```

FPR

CODE:

```
FPR = FP / float(FP + TN) * 100print('False Positive Rate: {:.4f}'.format(FPR))
```

OUTPUT:

```
   False Positive Rate: 16.4948
```

Specificity:

CODE:

```
specificity = TN /(TN+FP)*100print('Specificity : {0:0.4f}'.format(speci
ficity))
```

```
Specificity : 83.5052
```

CODE:

```
from sklearn.metrics import f1_score
```

```
print('F1_Score of Macro: ', f1_score(y_test, dt_pred)*100)
```

OUTPUT:

```
F1_Score of Macro:  74.13793103448276
```

CODE:

```
print("Micro Average f1 Score is: ", f1_score(y_test, dt_pred, average='
micro')*100)

print("Macro Average f1 Score is: ", f1_score(y_test, dt_pred, average='
macro')*100)

print("Weighted Average f1 Score is: ", f1_score(y_test, dt_pred, averag
e='weighted')*100

)print("f1 Score on Non Weighted score is: ", f1_score(y_test, dt_pred, a
verage=None)*100)
```

OUTPUT:

```
Micro Average f1 Score is:  80.51948051948051

Macro Average f1 Score is:  79.25646551724138

Weighted Average f1 Score is:  80.58595499328258

f1 Score on Non Weighted score is:  [84.375      74.13793103]
```

Classification Report of Decision Tree:

```
from sklearn.metrics import classification_report

print('Classification Report of Decision Tree: \n', classification_report(y_test, dt_pre
d, digits=4))
```

```
Classification Report of Decision Tree:

          precision    recall  f1-score   support



         0     0.8526    0.8351    0.8438        97
         1     0.7288    0.7544    0.7414       57



   accuracy                        0.8052      154

  macro avg     0.7907    0.7947    0.7926      154

weighted avg      0.8068     0.8052     0.8059   154
```

ROC Curve& ROC AUC

```
# Area under Curve:auc= roc_auc_score(y_test, dt_pred)

print("ROC AUC SCORE of Decision Treeis ", auc)
```

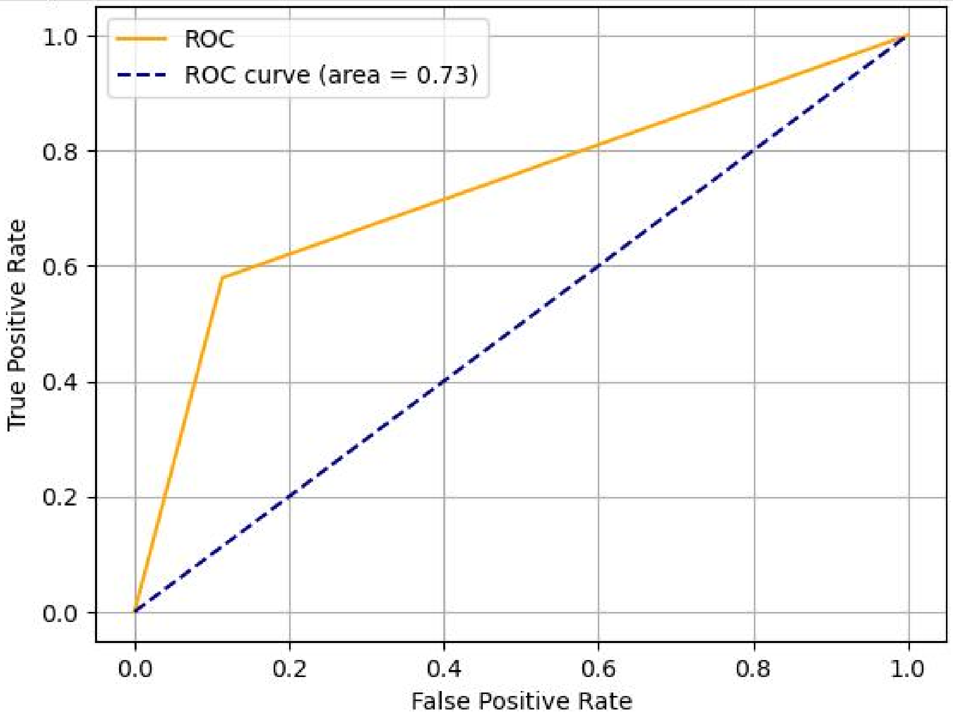
OUTPUT:

```
ROC AUC SCORE of Decision Treeis  0.7947187556520168
```

CODE:

```
from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as pltfpr, tpr, thresholds = roc_curve
(y_test, dt_pred)

plt.plot(fpr, tpr, color='orange', label="ROC")

plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='RO
C curve (area = %0.2f)' % auc(fpr, tpr))

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver Operating Characteristics (ROC) Curve of D
ecision Tree")

plt.legend()plt.grid()plt.show()
```

OUTPUT:



CONCLUSION:

In this project, we have imported the required libraries, dataset . We have done data cleaning, changing, replacing the null values and processed the data. We have used the given dataset link and performed RandomForest.