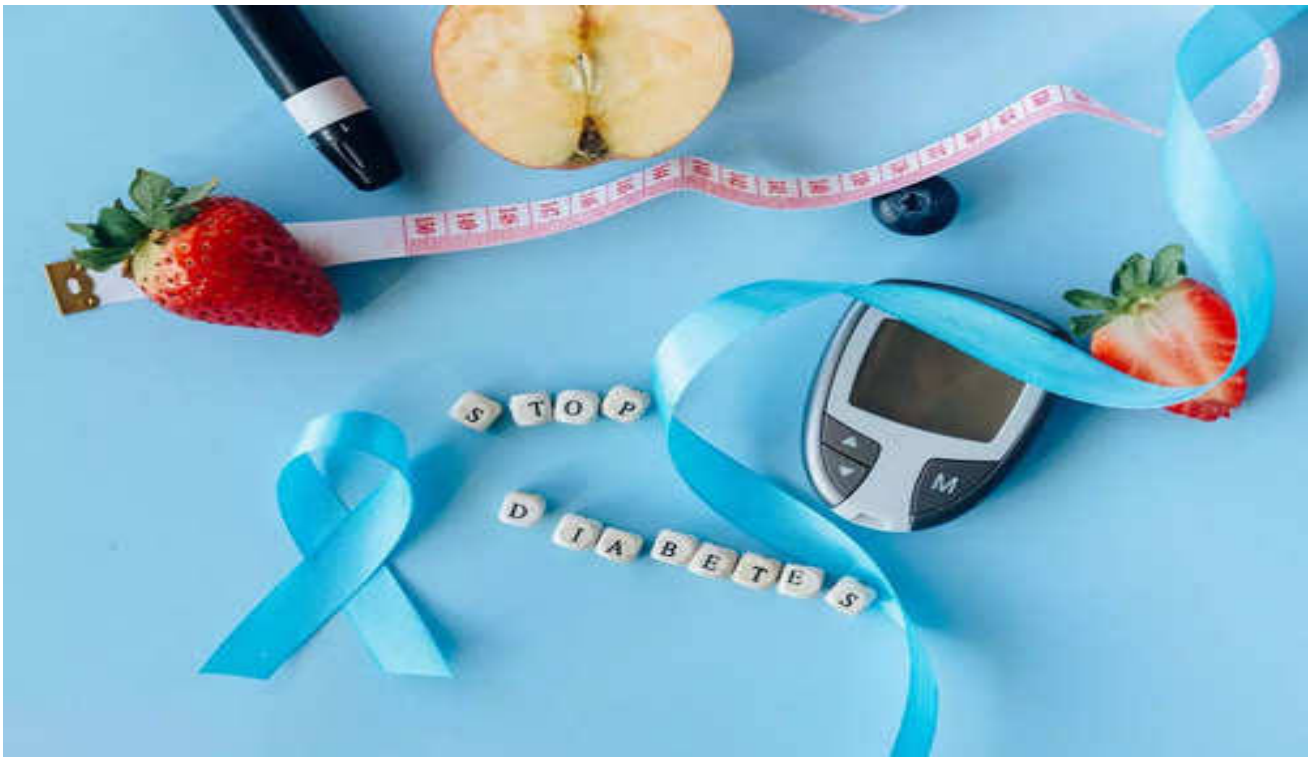


AI BASED DIABETES PREDICTION SYSTEM

S.SATHYA
311421104086
CSE-3rd Year

AI Based Diabetes Prediction System

Problem Definition

An AI-based diabetes prediction system is a valuable tool for identifying individuals at risk of developing diabetes or for helping manage the condition in those who already have it. Such a system typically relies on machine learning algorithms and data analysis to make predictions. Here's an overview of the steps involved in developing and deploying such a system to use :

Design Thinking

- Data collection
- Data Pre-processing
- Exploratory Data Analytics
- Feature Engineering
- Model Evaluation
- Deployment Selection

Data Collection:

Gather a comprehensive dataset containing medical records, patient demographics, lifestyle factors (e.g., diet, physical activity), family history, and clinical measurements (e.g., blood glucose levels, BMI) of individuals.

Data Preprocessing:

Clean and preprocess the dataset, handling missing values, outliers, and ensuring data consistency. Transform categorical variables into numerical representations if necessary.

Exploratory Data Analysis (EDA):

Conduct EDA to understand data distributions, relationships between features, and identify potential risk factors associated with diabetes. Visualization and statistical analysis will aid in feature selection.

Feature Engineering:

Create relevant features that can enhance the predictive capabilities of the model. This may involve deriving new features from existing data, considering interactions between variables, and selecting the most informative features.

Predictive Modelling with Logistic Regression:

Utilize logistic regression as the primary predictive modelling technique. Train the model on historical data to predict the probability of an individual developing diabetes based on selected features.

Model Evaluation:

Evaluate the logistic regression model using appropriate metrics, such as accuracy, precision, recall, F1-score, and the area under the Receiver Operating Characteristic (ROC-AUC) curve. Perform cross-validation to assess model generalization.

Deployment and Maintenance:

Determine the deployment strategy, such as integrating the model into electronic health records (EHR) systems, mobile applications, or web platforms.

Regularly update the model with new patient data to ensure its accuracy and relevance in predicting diabetes risk.

Implement data security measures and adhere to healthcare data regulations to protect patient privacy (e.g., HIPAA compliance).

Ethical Considerations:

Adhere to ethical standards and guidelines for healthcare AI, ensuring responsible handling of patient data and obtaining informed consent.

Implementation Procedure

Introduction

The goal of this document is to outline the steps for transforming the design of the AI-based diabetes prediction system, as previously defined, into an innovative solution.

Building a complete AI-based diabetes prediction system involves multiple steps, including obtaining the dataset, preprocessing the data, and building a predictive model. Below, I'll provide a step-by-step guide with code examples for each stage of the project.

Prerequisite step :

As you very well know, there are a quite a few programming languages (like Python, R) and tools that can be used to machine learning projects like this. But the most popular one is python and I going to use Python to do this project.

Initially, I will setup the project environment, here are the steps I will follow :

- Download **Miniconda** (an environment management system for installing and maintaining software packages), we use this to ease the process of installing necessary libraries for our project.
- Then I will create a repository and create a conda environment along with the necessary dependencies – **Pandas, NumPy, Matplotlib and Scikit.**

conda create --prefix ./env pandas numpy matplotlib scikit-learn

- Activate the environment and install Jupyter notebook – an IDE to run our project.

conda activate c:\Users\tvaru\Desktop\sample_project_ 1\env

- In **Jupyter notebook**, I will create a new python file and began the coding process.

conda installjupyter

Step 1: Obtaining the Dataset

The first step in any machine learning project is to obtain a dataset to work with. There are many online resources available to find the datasets suitable for our project. One such popular platform that hosts datasets is **Kaggle**. Since the goal of our project is to predict Diabetes, I search for diabetes patient's dataset, which is readily available as were already many such similar projects.

1. Go to the Kaggle dataset page you mentioned: Diabetes Data Set.
2. Click the "Download" button to get the dataset files.
3. Unzip the downloaded files to a directory on your local machine into your project repository.

This dataset has the following attributes : Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function and Age, It also contains the Outcome label which tells whether the patient has diabetes or not. Since there is a class label present by default, this dataset can be studied using Classification based algorithms.

Step 2: Data Preprocessing

Now that we have the dataset, we need to prepare it for analysis and modeling. This involves cleaning and organizing the data. Now, since I took this dataset from Kaggle, it is already fully pre-processed (i.e.) there are no missing values, all features have numerical values.

```
# Import necessary libraries

import pandas as pd

import numpy as np


# Load the dataset

data = pd.read_csv('diabetes_data.csv')


# Check the first few rows of the dataset

print(data.head())


# Check for missing values

print(data.isnull().sum())


# Split the data into features (X) and target variable (y)

X = data.drop('diabetes', axis=1)
```

```
y = data['diabetes']
```

In this step, we load the dataset, inspect the first few rows, and check for missing values. Since the data set contains the target class label, I can go ahead with classification or regression based approaches to build the model.

Step 3: Data Exploration (Optional)

Data exploration helps us understand the dataset better. We can create visualizations to gain insights into the data. We can visualize the data using the graphing library – **matplotlib**. I use the pairplot to understand the relationship between variables.

```
import matplotlib.pyplot asplt

import seaborn as sns


# Pairplot to visualize relationships between variables

sns.pairplot(data, hue='diabetes', diag_kind='kde')

plt.show()
```

Step 4: Data Splitting

The necessary step is to split the dataset into training and testing sets to train and evaluate our model. I use **Scikit's train_test_split** method to achieve this, the code for which as follows.

```
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets (80% train, 20% test)

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2,
random_state=42)
```

Step 5: Model Building (Logistic Regression)

Now, we'll build a logistic regression model to predict diabetes. As previously mentioned in the report, I am going to use Logistic Regression to build the model since

it is best suitable for datasets with class label, and also the class label has binary values, which in our case is true, since we have 0 or 1 for the Outcome.

The **Logistic Regression** method is found Scikit's linear_model sub module. After building and fitting the model to the training dataset, we predict the result of the testing dataset. And I evaluate the prediction using measures like accuracy, confusion matrix and classification report which standard metrics of classification based models.

```
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix


# Data preprocessing: Standardize features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Create and train the logistic regression model

model = LogisticRegression()

model.fit(X_train,y_train)


# Make predictions on the test set

y_pred = model.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test,y_pred)

confusion = confusion_matrix(y_test, y_pred)

report = classification_report(y_test, y_pred)


print(f'Accuracy: {accuracy}')
```

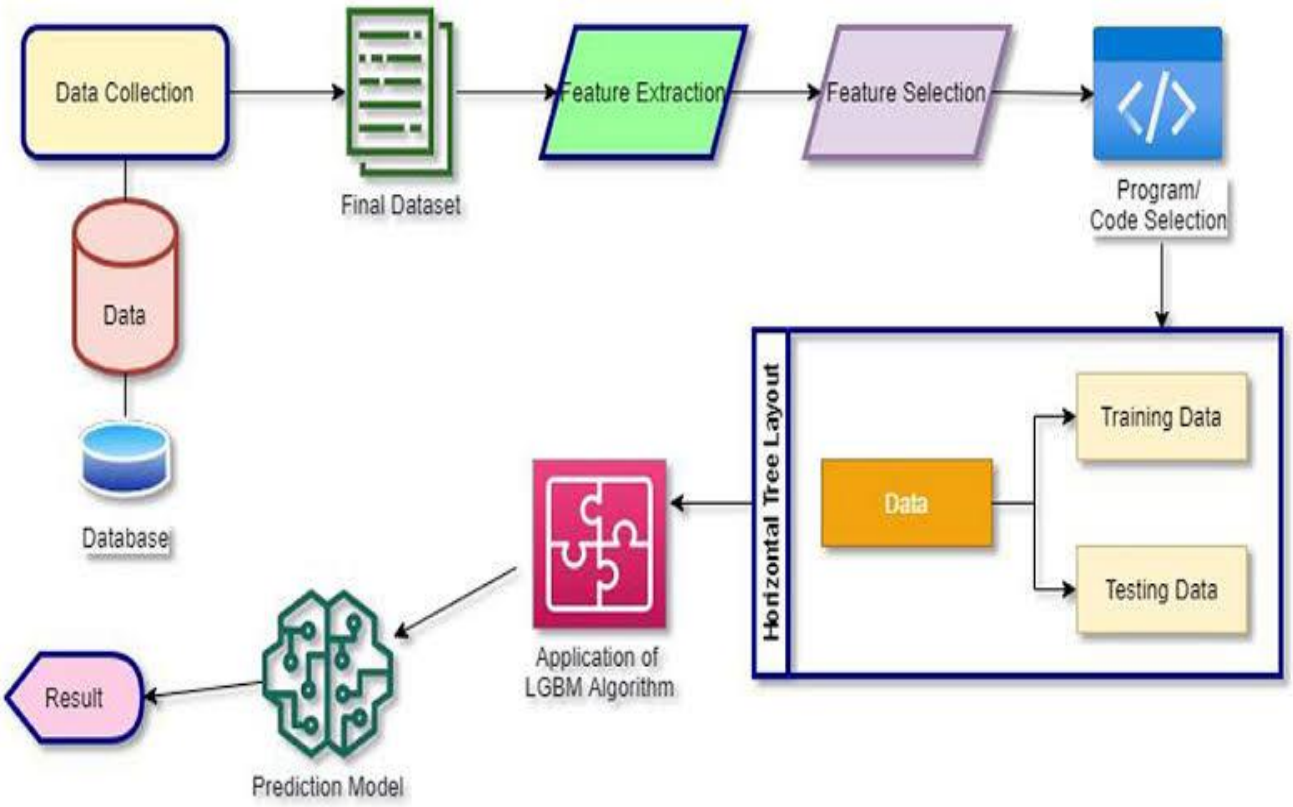
print(f'Confusion Matrix:\n{confusion}')

print(f'Classification Report:\n{report}')

Step 6: Model Deployment (Optional)

To deploy the model, you would need to integrate it into a healthcare system, application, or web platform, which involves additional coding and considerations beyond the scope of this explanation.

Diabetes Graphical representation:



Data collection and Preprocessing:

Gather a dataset containing relevant information about individuals, including features such as age, preganacies,BMI,insulin,blood pressure, and glucose levels. Datasets like the Diabetes Database can be useful.

Data splitting:

Split the dataset into training and testing sets to evaluate your model's performance.

Model Training:

Train the selected model on the training data using appropriate algorithms.

Model selection:

Choose an appropriate machine learning or deep learning model for diabetes prediction. Common models include logistic regression, decision trees or random forests.

Evaluation Performance:

Evaluate the model's evaluation in given diabetes database using

Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find kaggle Diabetes datasets in CSV format, but you can adapt this code to other formats as needed

To predict database using Diabetes

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
dataset = pd.read_csv('Documents/PS/diabetes.csv')
```

[4]:
```
dataset.head()
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

[5]:
```
dataset.shape
```

[5]:
```
(768, 9)
```

[6]:
```
dataset.info()
```

```
<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
```

[7]:
```
dataset.describe().T
```

[7]:

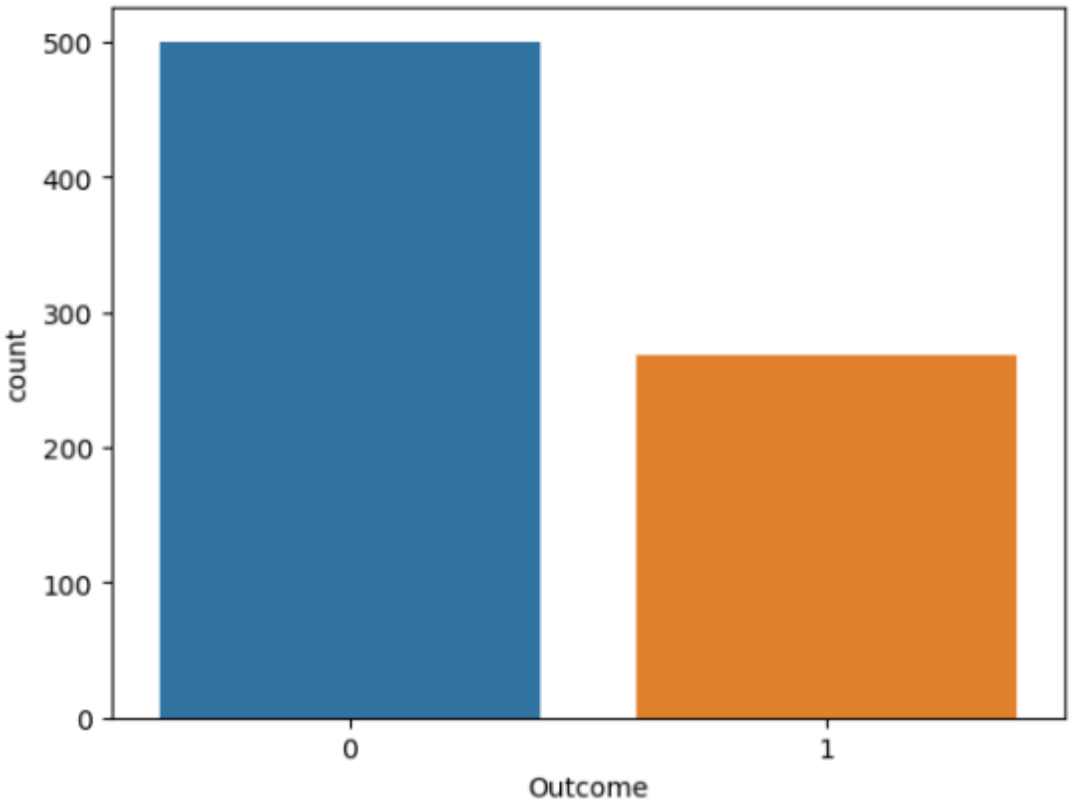
	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

[8]:
```
dataset.isnull().sum()
```

[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

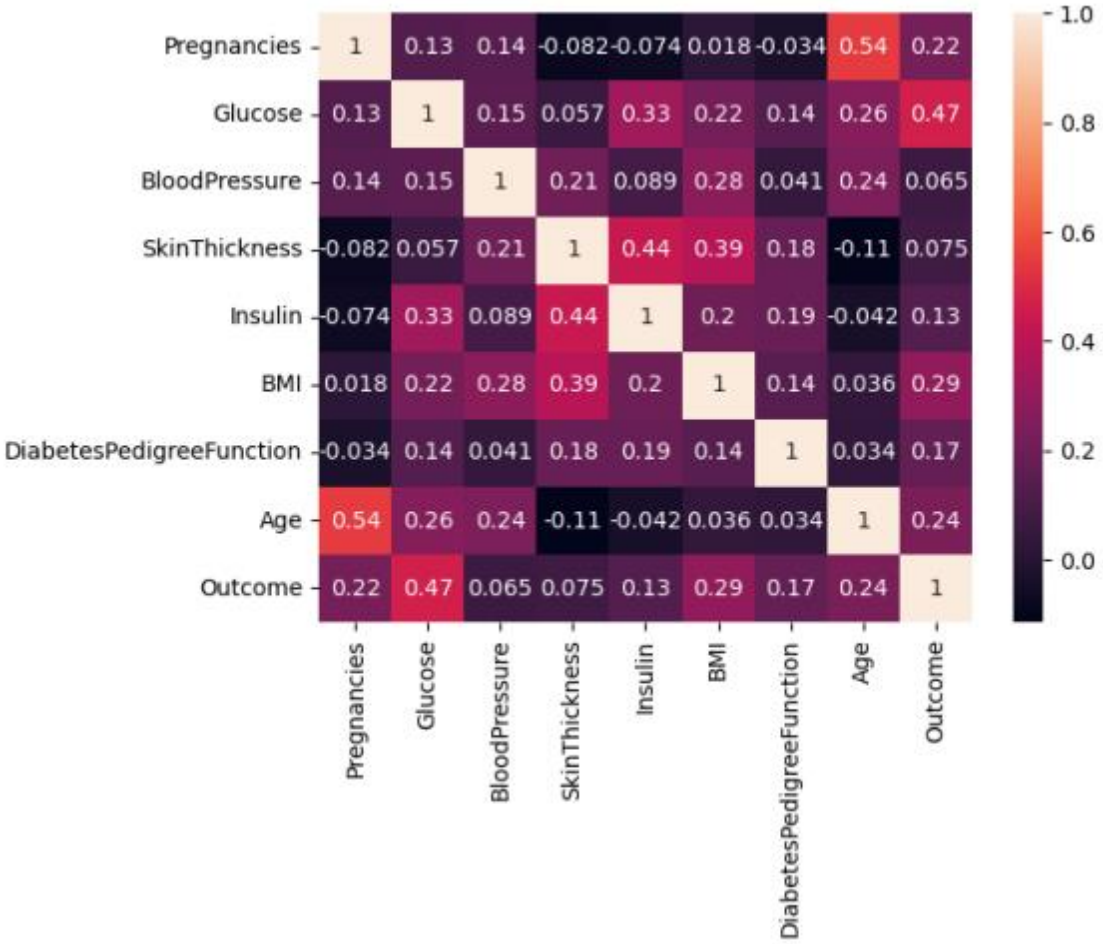
[9]:
```
sns.countplot(x = 'Outcome',data = dataset)
```

[9]: <Axes: xlabel='Outcome', ylabel='count'>



[12]:
```
# Pairplot
sns.pairplot(data = dataset, hue = 'Outcome')
plt.show()
```

[13]:
```
# Heatmap
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```



[14]:
```
# Replacing zero values with NaN
dataset_new = dataset
dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] =␣
 ↪dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].
 ↪replace(0, np.NaN)
```

[15]:
```
# Count of NaN
dataset_new.isnull().sum()
```

[15]:
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
```

```
Outcome                       0
dtype: int64
```

[16]:
```
# Replacing NaN with mean values
dataset_new["Glucose"].fillna(dataset_new["Glucose"].mean(), inplace = True)
dataset_new["BloodPressure"].fillna(dataset_new["BloodPressure"].mean(),␣
 ↪inplace = True)
dataset_new["SkinThickness"].fillna(dataset_new["SkinThickness"].mean(),␣
 ↪inplace = True)
dataset_new["Insulin"].fillna(dataset_new["Insulin"].mean(), inplace = True)
dataset_new["BMI"].fillna(dataset_new["BMI"].mean(), inplace = True)
```

[17]:
```
dataset_new.isnull().sum()
```

[17]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[18]:
```
y = dataset_new['Outcome']
X = dataset_new.drop('Outcome', axis=1)
```

[19]:
```
# Splitting X and Y
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 42, stratify = dataset_new['Outcome'] )
```

[20]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
y_predict = model.predict(X_test)
```

```
C:¥Users¥CSE_BAY4¥anaconda3¥Lib¥site-
packages¥sklearn¥linear_model¥_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
```

[21]:
```
y_predict
```

[21]:
```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

[22]:
```
# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_predict)
cm
```

[22]:
```
array([[85, 15],
       [25, 29]], dtype=int64)
```

[23]:
```
# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cm), annot=True)
```

[23]:
```
<Axes: >
```

Descision Tree:

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

[ ]:
```
DecisionTreeClassifier()
```

Making prediction:

Logistic Regression:

[ ]:
```
X_test.shape
```

[ ]:
```
(154, 8)
```

[ ]:
```
lr_pred=lr.predict(X_test)
```

[ ]:
```
lr_pred.shape
```

[ ]:
```
(154,)
```

Decision Tree:

[ ]:
```
dt_pred=dt.predict(X_test)
```

[ ]:
```
dt_pred.shape
```

[ ]:
```
(154,)
```

Model Evaluation for Logistic Regression:

Train Score and Test Score

[ ]:
```
# For Logistic Regression:
from sklearn.metrics import accuracy_score
print("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)
```

```
Train Accuracy of Logistic Regression:  77.36156351791531
Accuracy (Test) Score of Logistic Regression:  77.27272727272727
Accuracy Score of Logistic Regression:  77.27272727272727
```

```
[ ]: # 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)
```

```
Train Accuracy of Decesion Tree:  100.0
Accuracy (Test) Score of Decesion Tree:  80.51948051948052
Accuracy Score of Decesion Tree:  80.51948051948052
```

Confusion Matrix

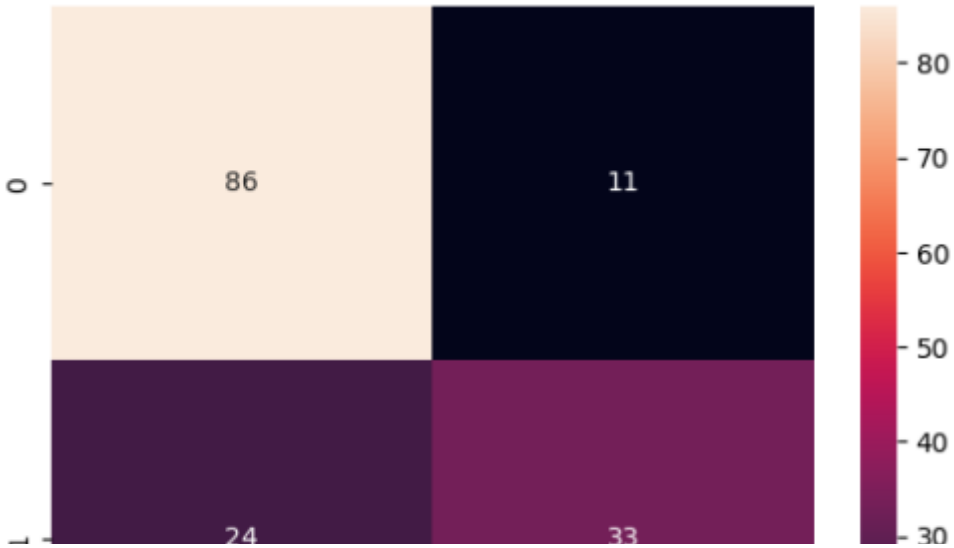
. *Confusion Matrix of "Logistic Regression"*

```
[ ]: from sklearn.metrics import classification_report, confusion_matrix

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

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

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

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



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

[ ]:
```
TN, FP, FN, TP
```

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

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = 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))
```

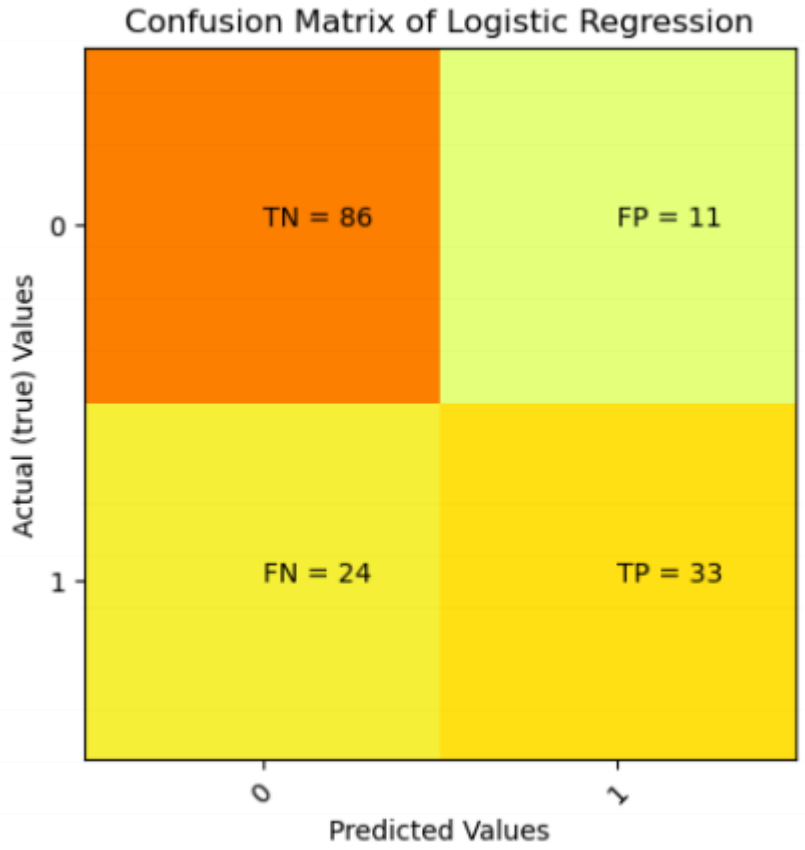
```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

[ ]:
```
77.27272727272727+22.727272727272727
```

[ ]:
```
100.0
```

[ ]:
```
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 Regression')
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()
```



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

[ ]:
```
col_0      0   1
Outcome
0         86  11
1         24  33
```

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

[ ]:
```
col_0       0   1  All
Outcome
0          86  11   97
1          24  33   57
```

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

```
F1_Score of Macro:  65.34653465346535
```

[ ]:
```
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:

[ ]:
```
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
```

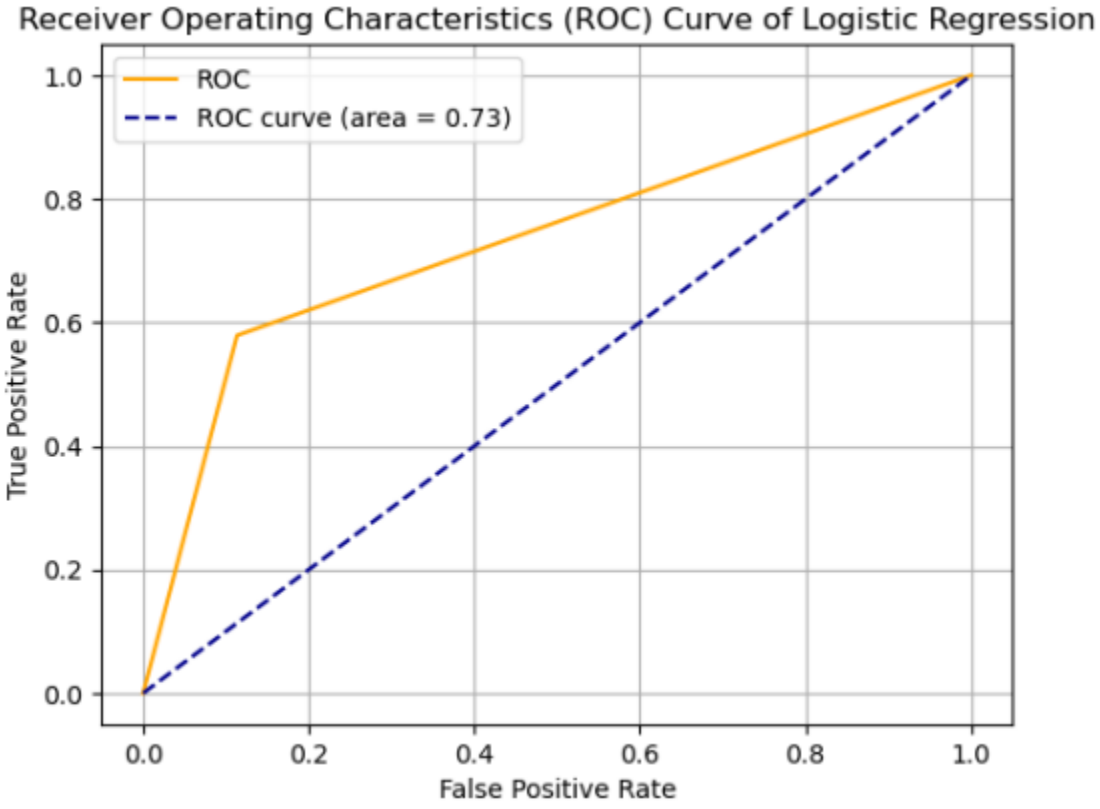
ROC Curve& ROC AUC

[ ]:
```
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

```
ROC AUC SCORE of logistic Regression is  0.7327726532826913
```

[ ]:
```
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='ROC 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 Logistic␣
↪Regression")
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrix:

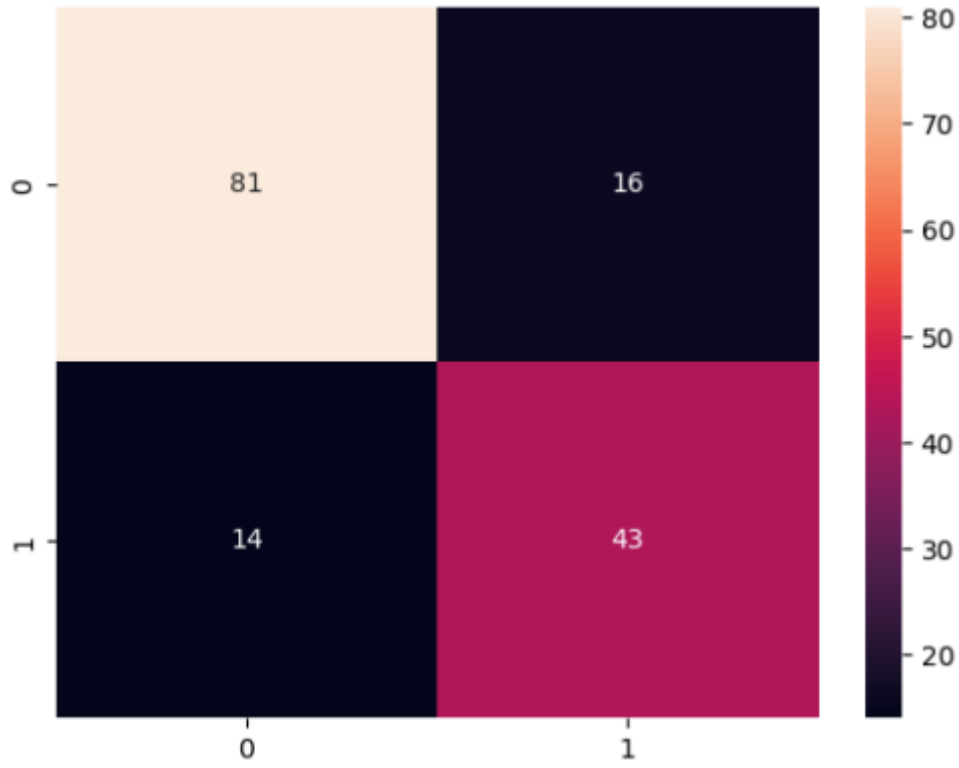
- Confusion matrix of "Decision Tree"

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix

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

[ ]:
```
array([[81, 16],
       [14, 43]])
```

[ ]:
```
sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

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



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

[ ]:
```
TN, FP, FN, TP
```

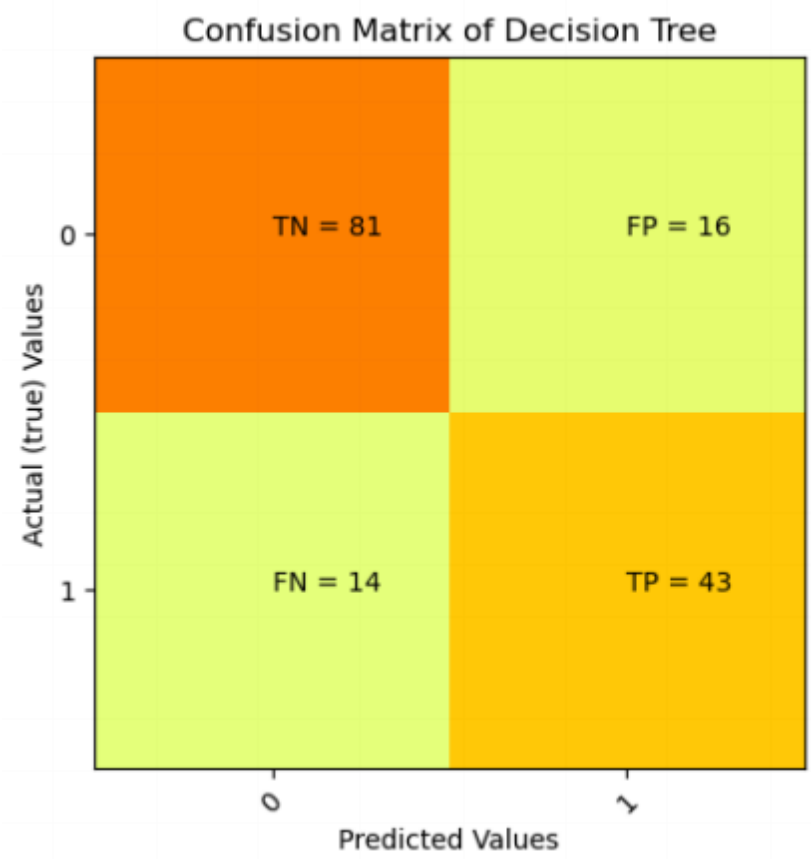
[ ]:
```
(81, 16, 14, 43)
```

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = 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))
```

```
TN - True Negative 81
FP - False Positive 16
FN - False Negative 14
TP - True Positive 43
Accuracy Rate: 80.51948051948052
Misclassification Rate: 19.480519480519483
```

[ ]:
```
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 Decision Tree')
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:

[ ]:
```
Precision score:

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

```
Precision Score: 72.8814
```

[ ]:
```
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)
```

```
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 72.88135593]
```

Recall:

```
[ ]: recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

```
recall_score 75.43859649122807
```

```
[ ]: from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, dt_pred)*100)
```

```
Recall or Sensitivity_Score: 75.43859649122807
```

```
[ ]: 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)
```

```
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 75.43859649]
```

FPR

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

```
False Positive Rate: 16.4948
```

Specificity:

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

```
Specificity : 83.5052
```

```
[ ]: from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, dt_pred)*100)
```

```
F1_Score of Macro:  74.13793103448276
```

```
[ ]: 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,␣
  ↪average='weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, dt_pred,␣
  ↪average=None)*100)
```

```
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_pred, digits=4))
```

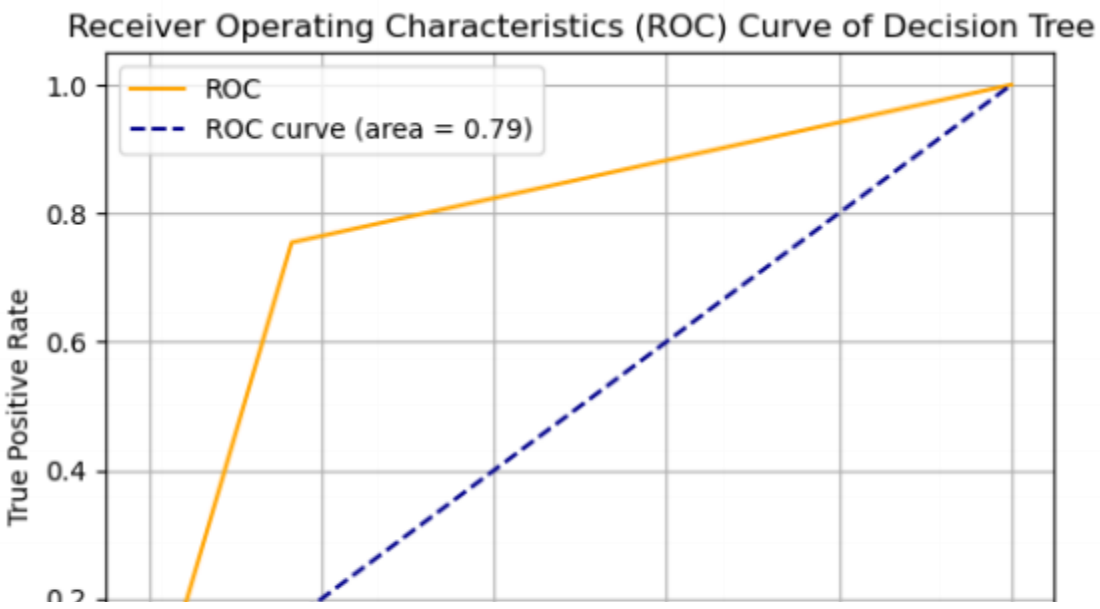
```
Classification Report of Decision Tree:
              precision    recall  f1-score   support

           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

```
[ ]: auc= roc_auc_score(y_test, dt_pred)
print("ROC AUC SCORE of Decision Treeis ", auc)
```

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

```
[ ]: from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, 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='ROC 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 Decision Tree")
plt.legend()
plt.grid()
plt.show()
```

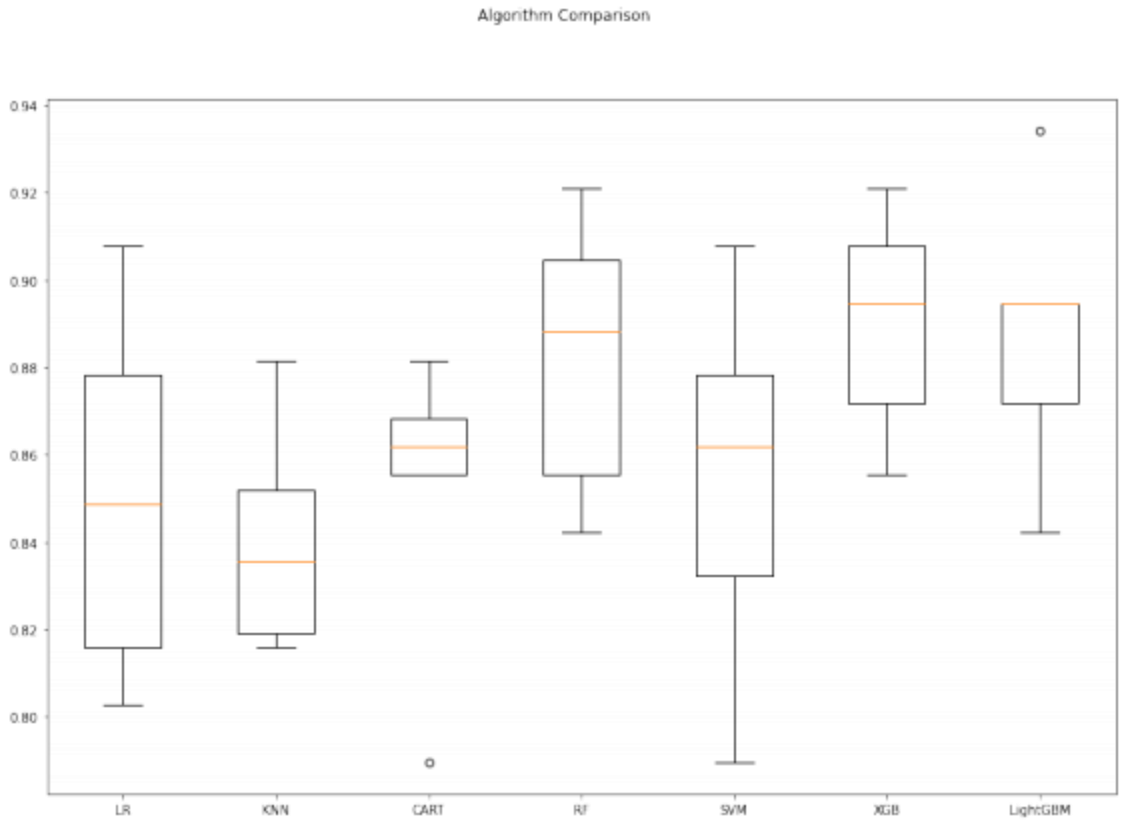


Base Models

```
[ ]: models = []
models.append(('LR', LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state = 12345)))
models.append(('RF', RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto', random_state = 12345)))
models.append(('XGB', GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM", LGBMClassifier(random_state = 12345)))

results = []
names = []
```

```
[ ]: for name, model in models:

        kfold = KFold(n_splits = 10, random_state = 12345)
        cv_results = cross_val_score(model, X, y, cv = 10, scoring= "accuracy")
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
LR: 0.848684 (0.036866)
KNN: 0.840789 (0.023866)
CART: 0.857895 (0.024826)
RF: 0.881579 (0.026316)
SVM: 0.853947 (0.036488)
XGB: 0.890789 (0.020427)
LightGBM: 0.885526 (0.024298)
```



5 Reporting

The aim of this study was to create classification models for the diabetes data set and to predict whether a person is sick by establishing models and to obtain maximum validation scores in the established models. The work done is as follows:

1) Diabetes Data Set read.

2) With Exploratory Data Analysis; The data set's structural data were checked. The types of variables in the dataset were examined. Size information of the dataset was accessed. The 0 values in the data set are missing values. Primarily these 0 values were replaced with NaN values. Descriptive statistics of the data set were examined.

3) Data Preprocessing section; df for: The NaN values missing observations were filled with the median values of whether each variable was sick or not. The outliers were determined by LOF and dropped. The X variables were standardized with the rubost method..

4) During Model Building; Logistic Regression, KNN, SVM, CART, Random Forests, XGBoost, LightGBM like using machine learning models Cross Validation Score were calculated. Later Random Forests, XGBoost, LightGBM hyperparameter optimizations optimized to increase Cross Validation value.

5) Result; The model created as a result of XGBoost hyperparameter optimization became the