**AI-Based Diabetes Prediction System**

**Phase 4 – Development Part 2**

**STEPS INVOLVED IN THIS PHASE:**

**1.** **Transforming the dataset to work with the model**

The first step is to transform the dataset to work with the model. This involves converting the categorical features to numerical features using dummy variables.

Dummy variables are a way of representing categorical features as numerical features. The get\_dummies function is used to convert the categorical features to dummy variables.

Next is to scale the numerical features. This is done using the RobustScaler class from scikit-learn. The RobustScaler class scales the features to a range of -1 to 1. This is used to the performance of the model by making it less sensitive to outliers.

**2. Training the model**

Next, the model is created and trained. The Random forest classifier is selected as the ensemble learning algorithm. Ensemble learning algorithms combine the predictions of multiple weak learners to create a more accurate and robust model.

Random forest classifiers work by building a large number of decision trees and averaging their predictions. Decision trees are used to split the data into smaller and smaller subsets based on the values of the features.

RandomForestClassifier class from scikit-learn is used to implement this. The RandomForestClassifier class takes a number of parameters, including the number of estimators.

The number of estimators specifies how many decision trees to build in the random forest. The random state makes sure that the model is reproducible, that is it will produce the same results each time it is trained on the same data.

**3. Assessing the performance of the model**

Next is to assess the performance of the model. To assess the model performance the following metrics are used accuracy, classification report, and confusion matrix.

The accuracy is the percentage of predictions that the model makes correctly.

The classification report gives the values for precision, recall, and F1-score.

The confusion matrix shows the number of predictions the model made for each class and the number of predictions that were correct.

**4. Prediction**

The final step is to make predictions on new data. The new data is passed to the model using the predict() method. The model will then return a prediction for each data point.

Three such sets of values were given and the result is obtained.

y\_pred1 predicts that the patient has a high probability of diabetes.

y\_pred2 predicts that the patient has a low probability of diabetes.

y\_pred3 predicts that the patient has a moderate probability of diabetes.

**IMPLEMENTATION:**

**1.Transforming the dataset to work with the model**

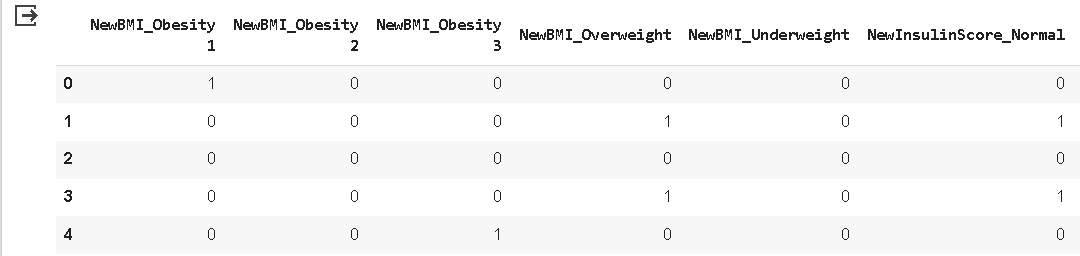
df = pd.get\_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], drop\_first = True)

categorical\_df = df[['NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

                     'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret']]

categorical\_df.head()

Output:



y = df["Outcome"]

X = df.drop(["Outcome",'NewBMI\_Obesity 1','NewBMI\_Obesity 2', 'NewBMI\_Obesity 3', 'NewBMI\_Overweight','NewBMI\_Underweight',

                     'NewInsulinScore\_Normal','NewGlucose\_Low','NewGlucose\_Normal', 'NewGlucose\_Overweight', 'NewGlucose\_Secret'], axis = 1)

cols = X.columns

index = X.index

from sklearn.preprocessing import RobustScaler

transformer = RobustScaler().fit(X)

X = transformer.transform(X)

X = pd.DataFrame(X, columns = cols, index = index)

X = pd.concat([X,categorical\_df], axis = 1)

**2.Training the model**

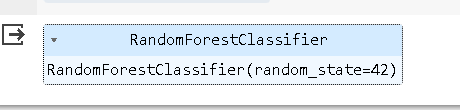
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the model on the training data

clf.fit(X\_train, y\_train)

Output:

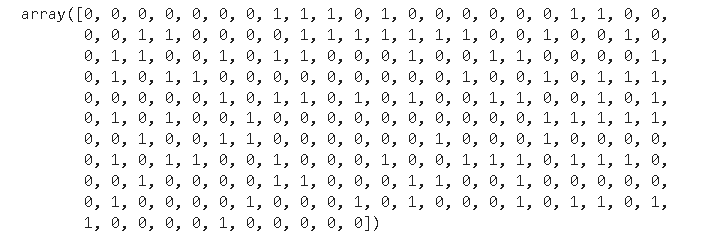


**3.Assessing the performance of the model**

y\_pred = clf.predict(X\_test)

y\_pred

Output:

****

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

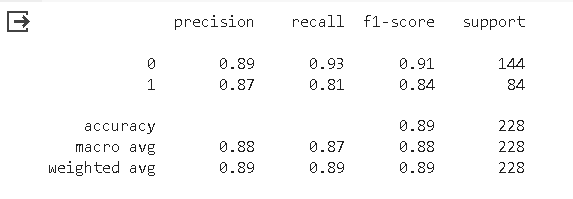
Output:



# Display other evaluation metrics

print(classification\_report(y\_test, y\_pred))

Output:



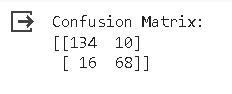
# Display the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print('Confusion Matrix:')

print(conf\_matrix)

Output:



**4.Prediction**

import numpy as np

X\_pred1=[0.6,0.770186,0.000,1.000000,1.000000,0.177778,0.672313,1.235294,1,0,0,0,0,0,0,0,0,1]

X\_pred1=np.array(X\_pred1).reshape(1,-1)

y\_pred1 = clf.predict(X\_pred1)

y\_pred1

Output:



import numpy as np

X\_pred2=[-0.4,-0.795031,-0.375,0.142857,0.000000,-0.600000,-0.600000,0.117647,0,0,0,1,0,1,0,1,0,0]

X\_pred2=np.array(X\_pred2).reshape(1,-1)

y\_pred2 = clf.predict(X\_pred2)

y\_pred2

Output:



import numpy as np

X\_pred3=[0.2,-0.546584,-0.5000,0.571429,1.000000,0.000000,-0.542020,0.117647,1,0,0,0,0,0,0,1,0,0]

X\_pred3=np.array(X\_pred3).reshape(1,-1)

y\_pred3 = clf.predict(X\_pred3)

y\_pred3

Output:

