**AI Based Diabetes Prediction System**

**Problem Definition:** The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

**Design thinking:**

* **Data collection:**  
  It is crucial for this project as the AI gets trained upon the dataset. More the quality of data, more is the prediction accuracy. Hence dataset is to be collected which should encompass a wide range of information, such as blood glucose levels, blood pressure readings, body mass index (BMI), and details on whether or not each person has been diagnosed with diabetes.
* **Data preprocessing:**

This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

* **Feature selection:**

Not all the data collected would be useful and effective for prediction and not all the data would contribute equally. Hence we need to select features which would affect the prediction the most to improve accuracy.

* **Model Selection:**

In order to perform tasks like classification and regression we would begin with Logistic Regression followed by complex models such as Random Forest, Gradient Boosting or Support Vector Machines. Scikit-learn in python would be used to implement these models.

* **Model Evaluation:**

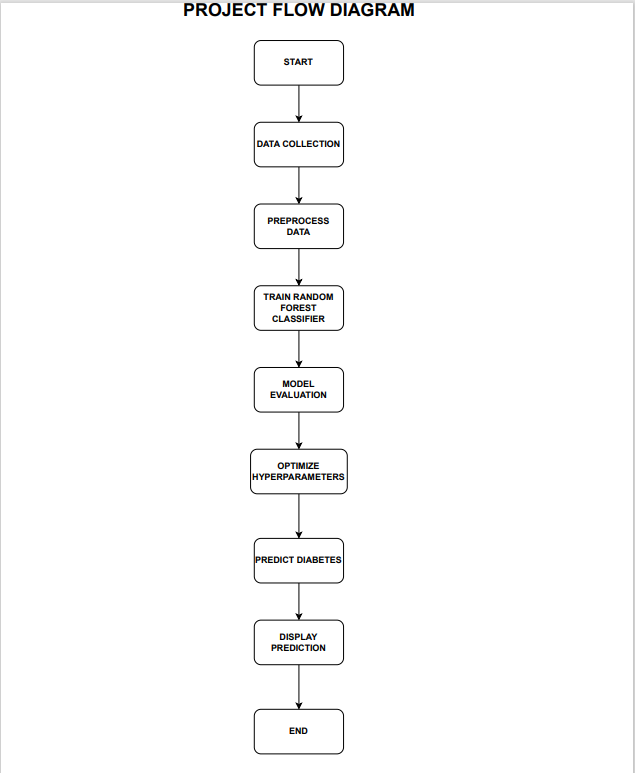
The performance of the model would be evaluated using approximate metrics like accuracy, precision, recall, F1-score and ROC-AUC. Visualization of the results would be done using ROC curves and confusion matrices.

* **Iterative improvement:**

We would continuously improve the performance and accuracy of our model by refining the data preprocessing steps. We would continuously strive to improve our approach with the suggestions given by mentors and domain experts.

* **Documentation and reporting:**

Documentation of the results play a crucial role in keeping track of all the results obtained. We would prepare a report that holds all the major results obtained. We would deploy our AI model in real-world healthcare setting so that the model would play an important part by contributing to the medical field.



**EXPLANATION:**

**1. Data Collection:**

In the data collection phase, we will gather relevant medical data from various sources. Information such as age, gender, blood pressure, body mass index (BMI), glucose levels, and family history of diabetes may be included in this data. It's critical to make sure the dataset is broad and thorough, reflecting a range of population risk factors and demographics. Because of this diversity, the model will be able to produce predictions that can be applied to a large number of people.

**2. Data Preprocessing:**

To guarantee the quality and appropriateness of the data for training the model, data preparation is an essential step. The tasks are broken out as follows:

**Managing Missing Values:** Medical datasets frequently contain missing values. In order to solve this, we will fill in the missing data points using the proper imputation techniques, such as mean or median imputation.

**Normalization and Standardization:** We will normalize the data to make sure that all features have comparable scales. This procedure stops the model from being dominated by one feature while it is being trained.

**Data division:** The dataset will be split into two sections: around 70% for training and 30% for testing. This allows us to train the model on one subset and evaluate its performance on another, ensuring that the model can generalize well to unseen data.

**3. Train Random Forest Classifier:**

We initialize a Random Forest model with particular hyperparameters in this stage. The number of trees in the forest, the greatest depth at which a tree can be found, and the bare minimum of samples needed in a leaf node are a few examples of these hyperparameters. Next, the training dataset is used to train the model. An effective option for this task is Random Forest, an ensemble learning technique that combines the predictions of several decision trees.

**4. Model Evaluation:**

To assess the model's performance, we use various evaluation metrics. These metrics include:

**Accuracy:** It measures the proportion of correct predictions.

**Precision:** It calculates the ratio of true positive predictions to the total positive predictions.

**Recall:** It computes the ratio of true positive predictions to the total actual positives.

**F1-Score:** It is the harmonic mean of precision and recall, providing a balanced measure of model performance.

Additionally, we use a confusion matrix to summarize true positives, true negatives, false positives, and false negatives, aiding in the interpretation of the model's performance. These metrics help us understand how well the model is at correctly classifying individuals as diabetic or non-diabetic.

**5. Optimize Hyperparameters:**

Despite the robustness of Random Forest, we can improve the model's performance by fine-tuning its hyperparameters. The basic settings for our particular dataset can be discovered by experimenting with various combinations of hyperparameters using methods like grid search or random search.

**6. Predict Diabetes:**

In this step, we gather new patient data, which includes the same features used for training the model. This data should be collected in a manner that mirrors the data format and quality used in the training dataset.

**7. Use the Trained Model to Predict Diabetes:**

The preprocessing steps for the new patient data will be the same as for the training data. We will use the trained Random Forest model to make predictions about the patient's likelihood of having diabetes after preprocessing.

**8. Display Prediction:**

The model's prediction is shown in the last step. This could include a clear indication of whether the patient is at risk for diabetes, as well as any additional information or recommendations based on the prediction.

**9. End:**

The procedure is complete once the prediction is shown, giving patients and healthcare providers useful details for early intervention and preventive measures.

**STEPS INVOLVED IN BUILDING THE PROJECT:**

**Step 1: Installing Required Libraries**

In this step, we import various Python libraries that will be used throughout our data preprocessing and analysis process. Each library has a specific purpose:

numpy and pandas are fundamental libraries for numerical operations and data manipulation, respectively. statsmodels is used for statistical modeling and hypothesis testing. seaborn and matplotlib are visualization libraries which are used to create meaningful plots and graphs. sklearn (Scikit-learn) is a powerful machine learning library, which offers tools for data preprocessing, modeling, and evaluation.

warnings module is used to manage warnings that might occur during the execution of our code.

**Step 2: Uploading the Dataset**

In this step, we use the pandas library to read a CSV file containing our dataset. The data set is saved in the variable df as a DataFrame, a two-dimensional labeled data structure with columns of various types.

**Step 3: Exploratory Data Analysis**

**Displaying the First Few Rows**

In this phase, the df.head() method is used to display the dataset's initial few rows. It displays the first few rows of actual data along with the column names.

Checking the Shape of the Dataset

Using df.shape, we obtain the dimensions of the dataset. It returns a tuple representing the number of rows and columns in the DataFrame.

**Dataset Information**

df.info() provides a concise summary of the DataFrame, including the total number of non-null entries in each column, the data type of each column, and memory usage information. This helps in understanding the structure and data types of the dataset.

**Step 4: Data Preprocessing**

**Handling Missing Values**

In this step, we handle the missing values in the dataset. Missing values can negatively impact the performance of machine learning models, so it is important to handle them appropriately.

**Replacing 0 with NULL to Identify Missing Value**

We replace zero values in specific columns (Glucose, BloodPressure, SkinThickness, Insulin, BMI) with NaN (Not a Number) to indicate that these are missing values.

**Missing Value Statistics**

We use df.isnull().sum() to count the number of missing values in each column. This provides information about which columns have missing data.

**Replacing Null Values with Median Value**

We substitute the median value for each column's missing values. This is a common technique to handle missing numerical data.

**Step 5: Data Preprocessing**

**Handling Outliers**

Outliers can impact statistical analyses and machine learning models. In this step, we identify and handle outliers in the dataset.

**Finding Outliers**

We use the Interquartile Range (IQR) method to detect outliers in each feature. If a feature has values outside the calculated bounds, it is considered to have outliers.

**Treating the Outliers**

We use the Local Outlier Factor (LOF) algorithm to detect and remove outliers. This method assigns an "outlier score" to each data point, and points with scores above a certain threshold are considered as outliers.

**Step 6: Selecting Relevant Features**

Feature selection is important for building accurate predictive models. In this step, we create new features based on existing ones.

Creating New Columns (NewBMI, NewInsulinScore, NewGlucose)

NewBMI: Categorizes BMI into different ranges (Underweight, Normal, Overweight, Obesity 1, Obesity 2, Obesity 3).

NewInsulinScore: Classifies insulin levels as either "Normal" or "Abnormal" based on defined thresholds.

NewGlucose: Categorizes glucose levels as "Low", "Normal", "Overweight", "Secret", or "High".

**Step 7: Final Dataset**

Finally, the shape of the dataset is displayed after all the preprocessing steps. This provides a summary of the dataset's transformation and shows which rows were eliminated as a result of the outlier treatment.

These steps in preparing the data guarantees that the data used to train and test models is of high quality and appropriate for precise predictions.

**IMPLEMENTATION:**

**INSTALLING LIBRARIES NEEDED FOR THE PROJECT:**

import numpy as np

import pandas as pd

import statsmodels.api as sm

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale, StandardScaler

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.metrics import confusion\_matrix, accuracy\_score, mean\_squared\_error, r2\_score, roc\_auc\_score, roc\_curve, classification\_report

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from lightgbm import LGBMClassifier

from sklearn.model\_selection import KFold

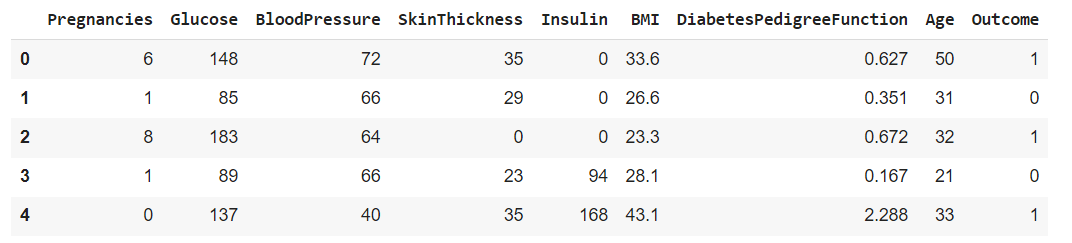
import warnings

warnings.simplefilter(action = "ignore")

**UPLOADING THE DATASET AND MAKING IT AS A DATAFRAME USING PANDAS:**

df = pd.read\_csv("/content/diabetes (1).csv")

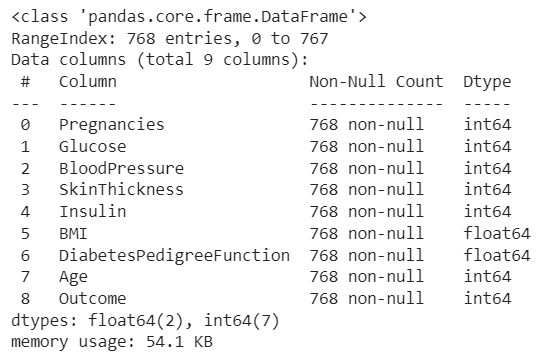
df.head()



df.shape



df.info()



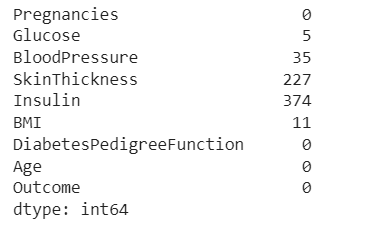
**DATA PREPROCESSING BY TREATING MISSING VALUES, OUTLIERS:**

#Replacing 0 with NULL to identify missing value

df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

#missing value statistics

df.isnull().sum()



#Replacing null values with median value

def median\_target(var):

    temp = df[df[var].notnull()]

    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset\_index()

    return temp

columns = df.columns

columns = columns.drop("Outcome")

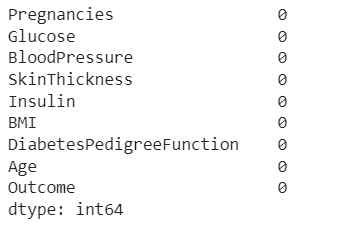
for i in columns:

    median\_target(i)

    df.loc[(df['Outcome'] == 0 ) & (df[i].isnull()), i] = median\_target(i)[i][0]

    df.loc[(df['Outcome'] == 1 ) & (df[i].isnull()), i] = median\_target(i)[i][1]

df.isnull().sum()



#To find which are the columns that has outliers

for feature in df:

    Q1 = df[feature].quantile(0.25)

    Q3 = df[feature].quantile(0.75)

    IQR = Q3-Q1

    lower = Q1- 1.5\*IQR

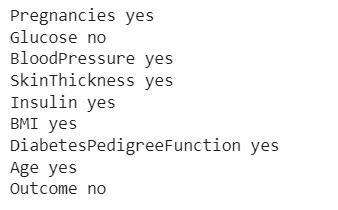
    upper = Q3 + 1.5\*IQR

    if df[(df[feature] > upper)].any(axis=None):

        print(feature,"yes")

    else:

        print(feature, "no")



#Treating the outliers

from sklearn.neighbors import LocalOutlierFactor

lof =LocalOutlierFactor(n\_neighbors= 10)

lof.fit\_predict(df)

df\_scores = lof.negative\_outlier\_factor\_

np.sort(df\_scores)[0:30]

threshold = np.sort(df\_scores)[7]

outlier = df\_scores > threshold

df = df[outlier]

#We see that the no.of rows is reduced as we have removed certain outliers

df.shape



**SELECTING RELEVANT FEATURES:**

#new column NewBMI to give range for BMI like underweight,overweight etc

NewBMI = pd.Series(["Underweight", "Normal", "Overweight", "Obesity 1", "Obesity 2", "Obesity 3"], dtype = "category")

df["NewBMI"] = NewBMI

df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]

df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9), "NewBMI"] = NewBMI[1]

df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9), "NewBMI"] = NewBMI[2]

df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9), "NewBMI"] = NewBMI[3]

df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9), "NewBMI"] = NewBMI[4]

df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]

#new column NewInsulinScore to tell whether the insulin level is normal or abnormal

def set\_insulin(row):

    if row["Insulin"] >= 16 and row["Insulin"] <= 166:

        return "Normal"

    else:

        return "Abnormal"

df = df.assign(NewInsulinScore=df.apply(set\_insulin, axis=1))

#new column NewGlucose to tell the level of glucose like normal,high etc

NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype = "category")

df["NewGlucose"] = NewGlucose

df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]

df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose[1]

df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucose[2]

df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]

#The no.of columns has increased as we have new features to the dataframe

df.shape



**Step 8:** **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.

**Step 9: 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.

**Step 10: 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.

**Step 11: 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**

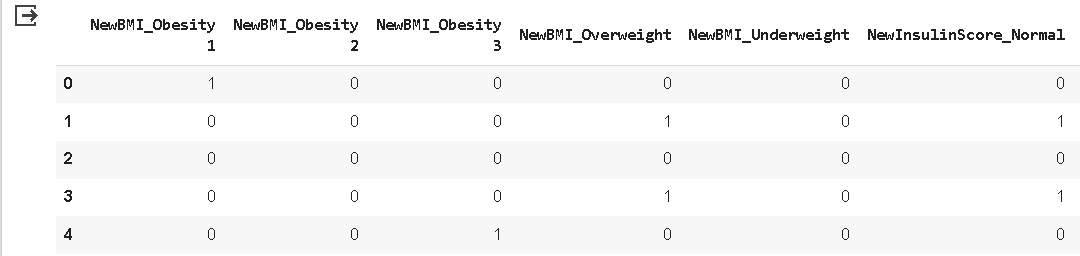
df = pd.get\_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], drop\_first = True)

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**

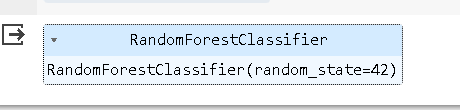
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

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

# Train the model on the training data

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

Output:

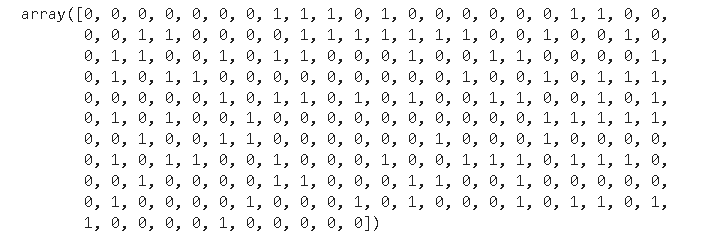


**3.Assessing the performance of the model**

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

y\_pred

Output:

****

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

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

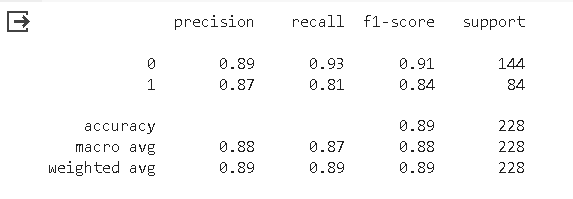
Output:



# Display other evaluation metrics

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

Output:



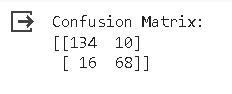
# Display the confusion matrix

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:

