# AI-Based Diabetes Prediction System

## Development phase -2 :

## Project overview :

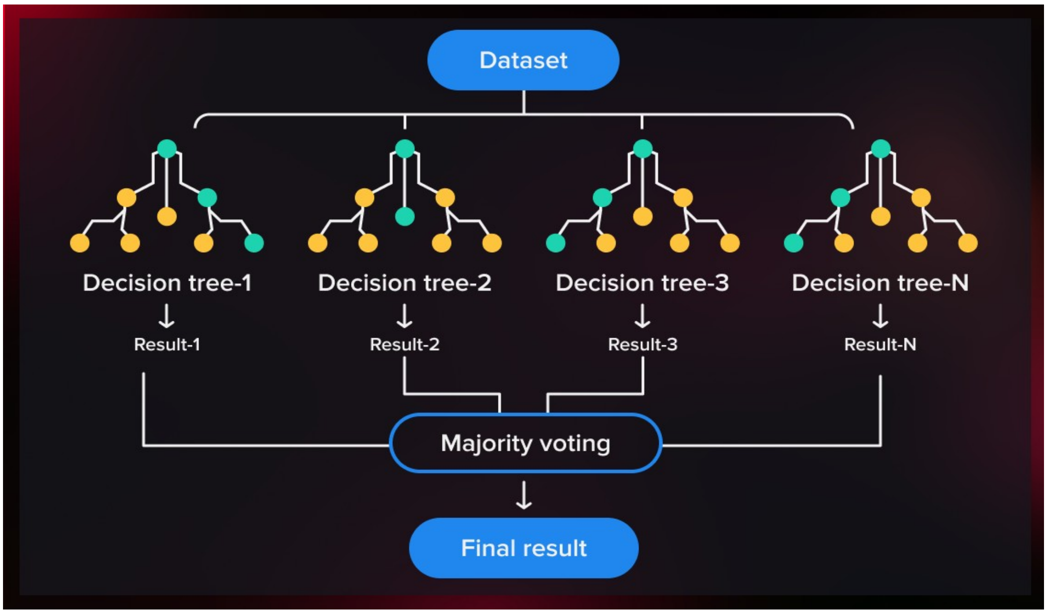
The goal of this project is to build an AI-based diabetes prediction system that can predict the likelihood of an individual developing diabetes based on a set of relevant features. This system will assist healthcare professionals in identifying individuals at risk of diabetes early, enabling them to take preventive measures.

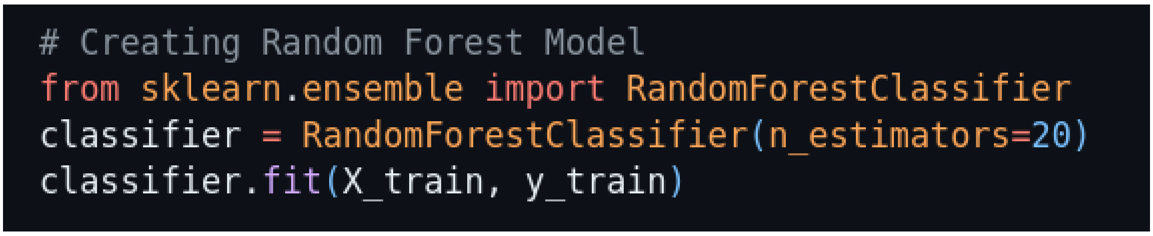
## 1.Data Collection and Preprocessing :

- I downloaded the dataset from kaggle .The given link is https://www.kaggle.com/datasets/mathchi/diabetes-data-set
- Gather a dataset containing historical patient data, including features such as age, BMI, family history, and glucose levels.
- Preprocess the data, handling missing values, outliers, and scaling where necessary using the python libraries .

## 2. Selecting a Machine Learning Algorithm :

- Choose an appropriate machine learning algorithm for diabetes prediction is the main task.
- The selected algorithm should be capable to handle the classification tasks and be suitable for medical data. So, i choose random forest from ski-kit learn to build my model.
- Random Forest is a versatile and powerful machine learning algorithm that is used for both classification and regression tasks.
- It is an ensemble learning method, which means it combines the predictions from multiple individual models to make more accurate and robust predictions.
- Random Forest is particularly popular and effective due to its ability to handle a wide range of data types and its ability to mitigate overfitting .





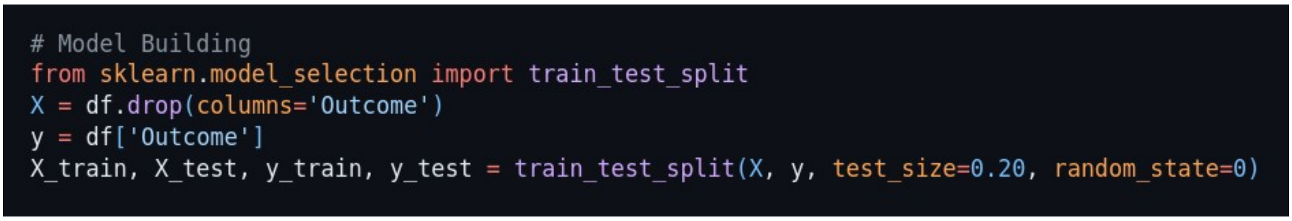
```
# Creating Random Forest Model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, y_train)
```

Import the random forest classifier to build my model using the RandomForestClassifier() .

Fit the model to the training data using `classifier.fit(X_train, y_train)`. This step is where the model learns from the training data.

## 3. Training the Model :

- Import the random forest classifier from ski-kit learn.
- Creates a RandomForestClassifier with 20 decision trees (n_estimators=20).
- Train the machine learning model using the training data.
- Optimize hyperparameters for improved performance.
- Fits the classifier to the training data (X_train, y_train) to train the model.
- We start building the model by first splitting the dataset into features (X) and the target variable (y).
- We then split the data into training (X_train, y_train) and testing (X_test, y_test) sets using the `train_test_split` function from scikit-learn.
- We choose a Random Forest Classifier with 20 estimators to build our model.



```
# Model Building
from sklearn.model_selection import train_test_split
X = df.drop(columns='Outcome')
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

## 4. Evaluating Model Performance :

- Evaluate the model's performance using appropriate evaluation metrics.
- Visualize the results to gain insights into model performance.
- Make necessary adjustments to the model if its performance is not satisfactory.
- Evaluate the model's performance using appropriate metrics such as accuracy, precision, recall, F1 score, or ROC AUC.

- If the model's performance is not satisfactory, you can fine-tune the hyperparameters, such as the number of trees ('n_estimators'), maximum depth of trees ('max_depth'), and more.

### Accuracy -

Calculate the accuracy to determine the percentage of correctly predicted instances.

```
from sklearn.metrics import accuracy_score
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

### Confusion Matrix -

Examine the confusion matrix to understand the model's true positive, true negative, false positive, and false negative predictions.

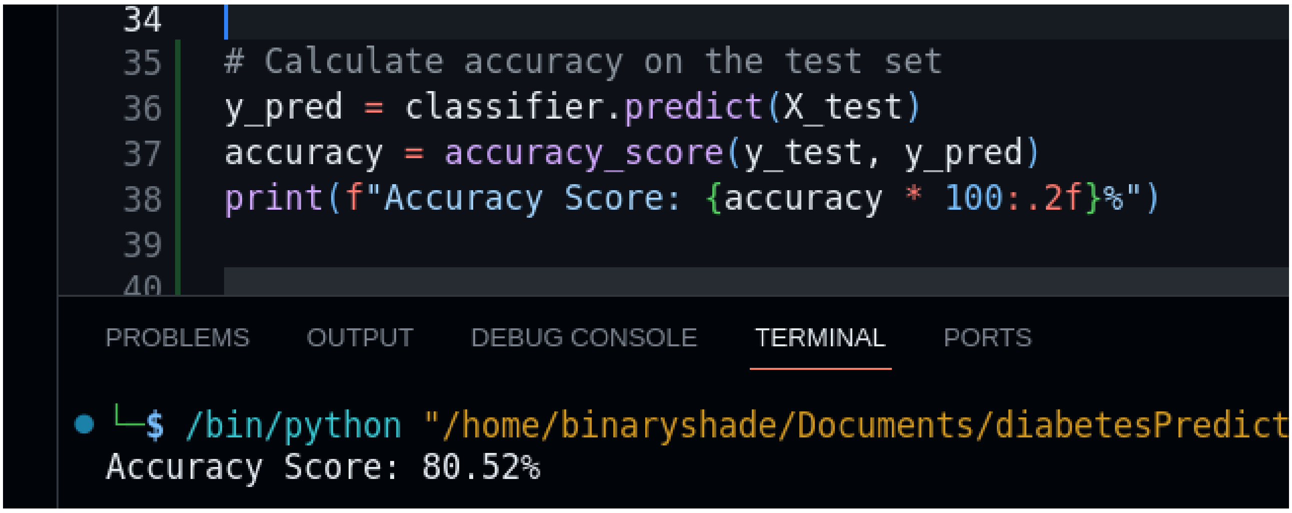
```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

### Precision, Recall, and F1-Score:

Calculate precision, recall, and F1-score to evaluate the model's performance on positive class predictions.

```
from sklearn.metrics import precision_score, recall_score, f1_score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-Score: {f1:.2f}')
```

I evaluated the model using the following evaluation methods .



```
# Calculate accuracy on the test set
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy * 100:.2f}%")
```

```
└─$ /bin/python "/home/binaryshade/Documents/diabetesPredict
Accuracy Score: 80.52%
```

My model scored with a accuracy of 80.2% .

**diabetes-prediction.py -**

```
# Importing essential libraries
import numpy as np
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Loading the dataset
df = pd.read_csv('diabetes.csv')

# Renaming DiabetesPedigreeFunction as DPF
df = df.rename(columns={'DiabetesPedigreeFunction': 'DPF'})

# Replacing the 0 values from ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI'] by NaN
df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']] = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']].replace(0, np.NaN)

# Replacing NaN values by mean, median depending upon distribution
df['Glucose'].fillna(df['Glucose'].mean(), inplace=True)
df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace=True)
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace=True)
df['Insulin'].fillna(df['Insulin'].median(), inplace=True)
df['BMI'].fillna(df['BMI'].median(), inplace=True)

# Model Building
X = df.drop(columns='Outcome')
y = df['Outcome']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# Creating Random Forest Model
classifier = RandomForestClassifier(n_estimators=20)
classifier.fit(X_train, y_train)

# Calculate accuracy on the test set
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy * 100:.2f}%")

# Creating a pickle file for the classifier
filename = 'diabetes-prediction-rfc-model.pkl'
pickle.dump(classifier, open(filename, 'wb'))
```