**AI\_PHASE5**

**Diabetes Prediction System using AI**

**Development Process Overview:**

**Problem Statement:**

* To develop an AI-powered diabetes prediction system that utilizes machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes.

**Design Thinking Process:**

* Define the problem and user needs: Identify the target population, data sources, and desired outcomes for the diabetes prediction system.
* Ideation and prototyping: Brainstorm AI models, data preprocessing techniques, and evaluation metrics, then create a prototype to test the approach.
* User testing and iterative development: Gather feedback from healthcare professionals and patients, refine the system, and continuously improve its accuracy and usability.

**Phases of development :**

* Firstly, we used a defined dataset that included the required parameters of the patients. Then for Data Pre-process, we Imputed missing values, removed outliers, and standardized the data to make it suitable for modeling.
* Secondly, we used domain knowledge and feature selection techniques to choose the most important attributes or engineer new features.
* Then we regularized the model, split the data into training and validation sets, and used cross-validation to find the best hyperparameters. We also used various tools to conduct accuracy tests of our results and visualized them.
* Finally we were able to provide clear information about the system's limitations, benefits, and how to interpret the results.

**Dataset used:**

We have used a diabetes dataset provided in kaggle

<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

The dataset has:

Information about People: Things like age, gender, weight, and family history.Health Numbers: Measurements like blood sugar, cholesterol, and blood pressure.Diabetes Status: It shows if a person has diabetes or not.

**Machine learning algorithm used:**

* In the development of our diabetes prediction system, we employed a Machine Learning process with the Random Tree Classifier algorithm.We opted for this algorithm as it provided one of the best results with 80.5% accuracy score.

**Model Training and Evaluation:**

Our approach involved extensive model training and evaluation.

For assessment, we utilized key performance metrics such as:

* the confusion matrix to analyze classification results,
* and ROC (Receiver Operating Characteristic) graphs to visualize the model's accuracy and its ability to discriminate between positive and negative cases.

This comprehensive evaluation framework ensured the robustness and reliability of our diabetes prediction system.

**Project Final Submission :**

**Diabetes Prediction System using AI**

**Problem Statement:**

There is a pressing need for an AI-based diabetes prediction system to address the growing global burden of diabetes. Despite advancements in healthcare, diabetes remains a major public health concern, with millions of people at risk of developing the condition. Current diagnostic methods are often reactive and lack precision, leading to delayed interventions and increased healthcare costs. An AI-based prediction system is required to accurately identify individuals at risk of diabetes, allowing for early interventions, personalized care, and improved healthcare resource allocation. Such a system should leverage data from various sources, including medical records, lifestyle factors, and genetic information, to provide accurate and timely predictions, ultimately reducing the prevalence and impact of diabetes on individuals and healthcare systems.

**Steps Involved with code:**

**1.Data gathering and analysis:**

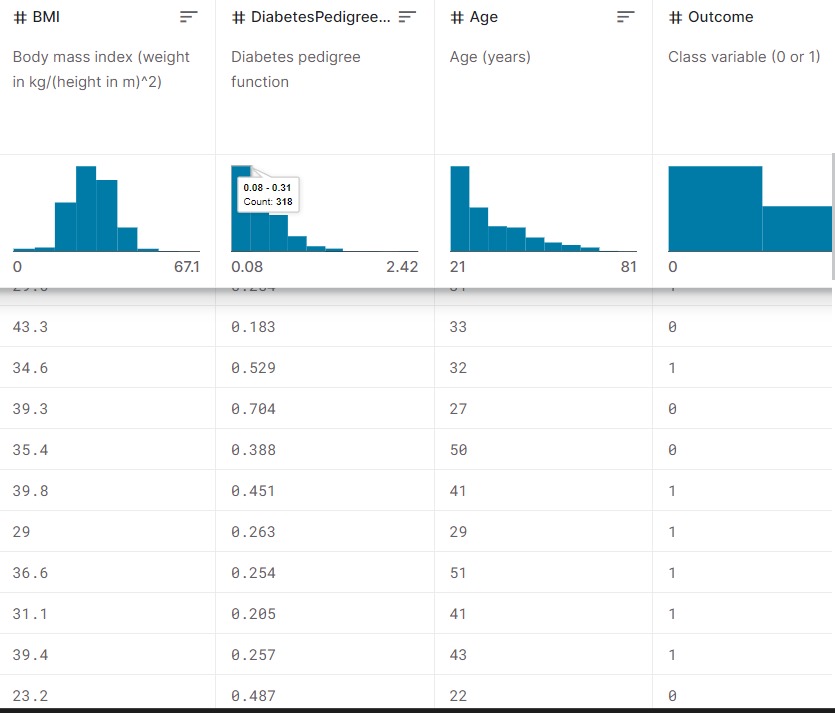
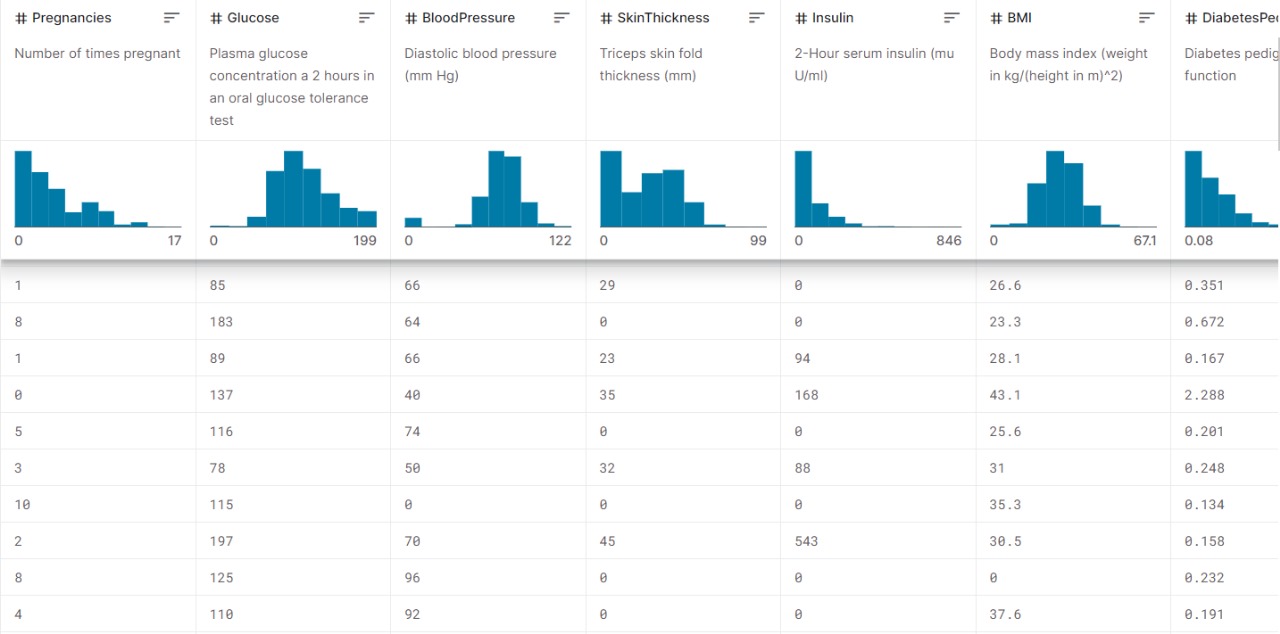
Developing a predictive system for diabetes involves utilizing datasets that contain various health-related attributes and information about individuals. These datasets are fundamental for training machine learning models to predict the likelihood of an individual having or developing diabetes based on certain characteristics.The dataset consists of a collection of information from a group of individuals, both diabetic and non-diabetic, with various attributes and health-related features.Attributes typically include biological, demographic, and clinical information such as age, gender, BMI (Body Mass Index), blood pressure, insulin levels, glucose levels, family history, and other relevant medical indicators.

Each row in the dataset represents a unique individual, and the columns contain different attributes and features that describe these individuals.

The dataset may have been curated from medical records, clinical trials, or research studies related to diabetes.

Dataset link: <https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

Dataset Screenshots:



We’ll start with importing Pandas and NumPy into our python environment and loading a .csv dataset into a pandas dataframe named df. To see the first five records from the dataset we use pandas df.head() function. We’ll also use seaborn and matplotlib for visualization. Each and every examples shown in this article are verified on a Jupyter notebook.

In[1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

In[2]:

#importing dataset

df = pd.read\_csv('../input/pima-indians-diabetes-database/diabetes.csv')

df.head()

Out[2]:

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

The dataset contains 768 observable with eight feature variables and one target variable. Before starting to analyze the data and draw any conclusions, it is essential to understand the presence of missing values in any dataset. To do so the simplest way is to use df.info() function which will provide us the column names with the number of non-null values in each column.

In[3]:

df.dtypes

Out[3]:

Pregnancies int64

Glucose int64

BloodPressure int64

SkinThickness int64

Insulin int64

BMI float64

DiabetesPedigreeFunction float64

Age int64

Outcome int64

dtype: object

In[4]:

df.info()

Out[4]:

<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

According to the output we don’t observe any null values. But there are five features such as Glucose, BloodPressure, SkinThickness, Insulin and BMI contains zero values which is not possible in the medical history. We will consider these values as missing values.We’ll replace the zero values

to NaN and then impute them with their mean value.

In[5]:

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

In[6]:

# making a list of columns with total number of missing values

print('Column'+ '\t\t\t\t Total missing Values'+'\t\t\t\t % of missing values')

#print("\n")

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

Column Total missing Values % of missing values

Pregnancies 0 0.00

Glucose 5 0.65

BloodPressure 35 4.56

SkinThickness 227 29.56

Insulin 374 48.70

BMI 11 1.43

DiabetesPedigreeFunction 0 0.00

Age 0 0.00

Outcome 0 0.00

In[7]

df['Glucose'].fillna(df['Glucose'].mean(), inplace=True)

df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace=True)

df['SkinThickness'].fillna(df['SkinThickness'].mean(), inplace=True)

df['Insulin'].fillna(df['Insulin'].mean(), inplace=True)

df['BMI'].fillna(df['BMI'].mean(), inplace=True)

In[8]

# making a list of columns with total number of missing values

print('Column'+ '\t\t\t\t Total missing Values'+'\t\t\t\t % of missing values')

#print("\n")

for i in df.columns:

print(f"{i: <50}{df[i].isnull().sum():<30}{((df[i].isnull().sum())\*100)/df.shape[0]: .2f}")

Out[8]:

Column Total missing Values % of missing values

Pregnancies 0 0.00

Glucose 0 0.00

BloodPressure 0 0.00

SkinThickness 0 0.00

Insulin 0 0.00

BMI 0 0.00

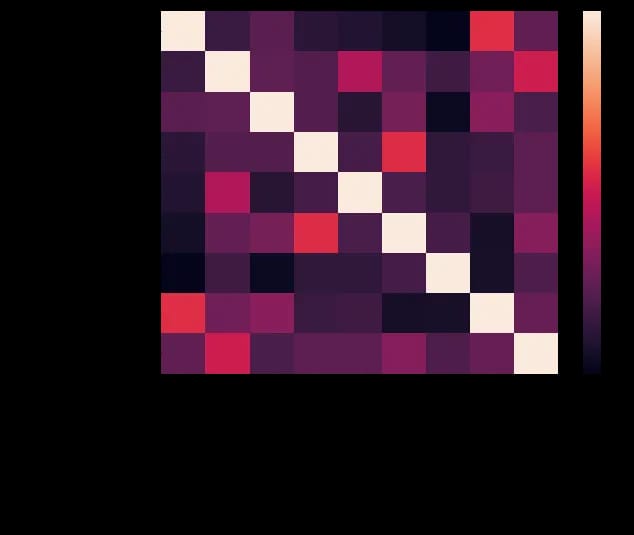
DiabetesPedigreeFunction 0 0.00

Age 0 0.00

Outcome 0 0.00

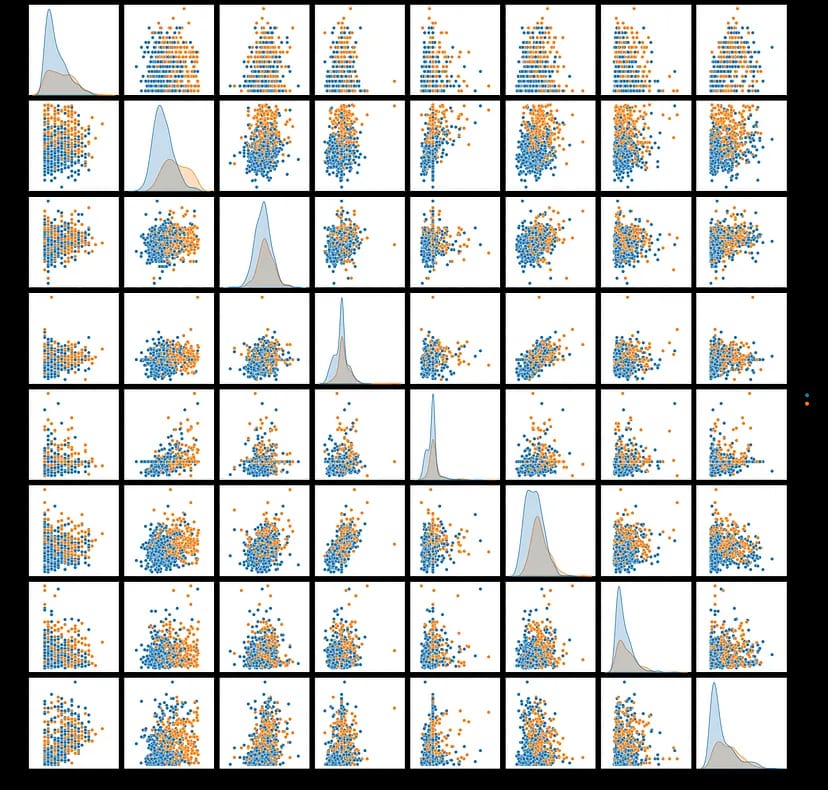
**2.Data Visualization**

The correlation between each columns are visualized using heatmap. From the output, the lighter colors indicate more correlation. We notice the correlation between pairs of features, like age and pregnancies, or BMI and skin thickness, etc.



To plot pairwise relationships in a dataset we use sns.pairplot() function and labeled the datapoints based on the target variable classes..

sns.pairplot(df,hue=’Outcome’)



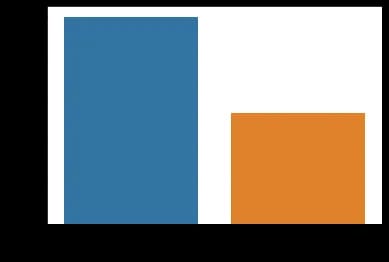
**3.Classification**

We need to separate the dataset into features and target variables. Following the popular convention, we call the dataframe with feature variables as X and the one with target variable as y.

X=df.drop('Outcome',axis=1)

y=df['Outcome']

Let’s visualize the target variable and have a look at how many people in the dataset are diabetic and how many are not.



Using sklearn’s train\_test\_split, we split the feature (X) and target (y) dataframes into a training set (80%) and testing set (20%). Training set is used for building classification model and testing set is used for evaluating the performance of the model.

from sklearn.metrics import confusion\_matrix

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

plt.figure(figsize=(7, 5))

sns.heatmap(mat, annot=True)

Before implementing classification algorithm, we scale the feature variables of our dataset using sklearn’s StandardScaler() function. This function standardize the features by removing the mean and scaling to unit variance.

from sklearn.preprocessing import StandardScaler

scaling\_x=StandardScaler()

X\_train=scaling\_x.fit\_transform(X\_train)

X\_test=scaling\_x.transform(X\_test)

**4.Training and Evaluating Model**

We’ll be using a machine simple learning model called Random Forest Classifier. We train the model with standard parameters using the training dataset. The trained model is saved as “rcf”. We evaluate the performance of our model using test dataset. Our model has a classification accuracy of 80.5%.

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()

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

rfc.predict(X\_test)

rfc.score(X\_test, y\_test)

Output:

0.8051948051948052

**5.Plotting decision boundaries**

A decision boundaries plot works well only with two features. Our data has eight features, but we still can plot decision boundaries by choosing which features to use. We plot decision boundary for each two possible features and see how well the model classifies the patients.

from mlxtend.plotting import plot\_decision\_regions

def classify\_with\_rfc(X,Y):

x = df[[X,Y]].values

y = df['Outcome'].astype(int).values

rfc = RandomForestClassifier()

rfc.fit(x,y)

# Plotting decision region

plot\_decision\_regions(x, y, clf=rfc, legend=2)

# Adding axes annotations

plt.xlabel(X)

plt.ylabel(Y)

plt.show()

feat = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']

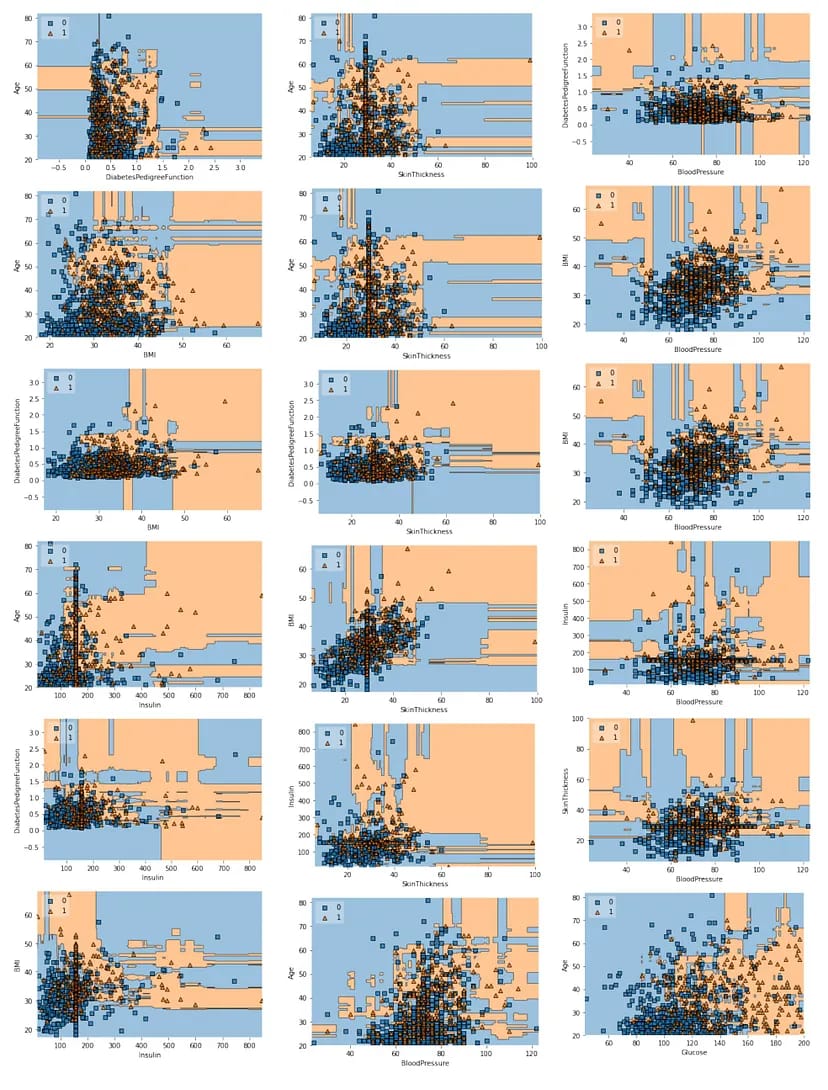
size = len(feat)

for i in range(0,size):

for j in range(i+1,size):

classify\_with\_rfc(feat[i],feat[j])

NB: 0 — Non Diabetic and 1 — Diabetic



The distributions shows our model classifies the patients really well. For a detailed evaluation of our model, we look at the confusion matrix.

from sklearn.metrics import confusion\_matrix

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

plt.figure(figsize=(7, 5))

sns.heatmap(mat, annot=True)

from sklearn.metrics import classification\_report

target\_names = ['Diabetes', 'Normal']

print(classification\_report(y\_test, y\_pred, target\_names=target\_names))

Output:

precision recall f1-score support

Diabetes 0.86 0.86 0.86 107

Normal 0.68 0.68 0.68 47

accuracy 0.81 154

macro avg 0.77 0.77 0.77 154

weighted avg 0.81 0.81 0.81 154

**6.ROC curve**

from sklearn.metrics import roc\_curve

y\_pred\_proba = rfc.predict\_proba(X\_test)[:,1]

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba

plt.plot([0,1],[0,1],'k-')

plt.plot(fpr,tpr, label='Knn')

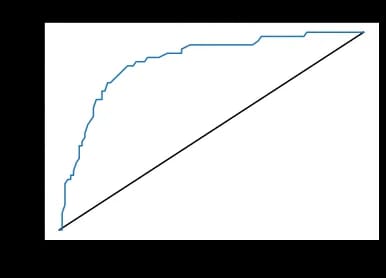
plt.xlabel('fpr')

plt.ylabel('tpr')

plt.title('ROC curve')

plt.show()

**Output:**



from sklearn.metrics import roc\_auc\_score

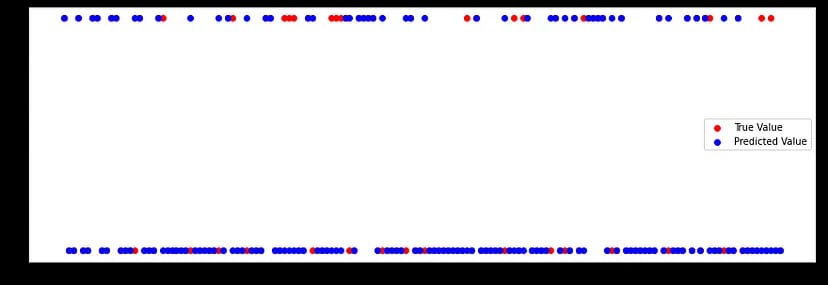
roc\_auc\_score(y\_test,y\_pred\_proba)

Output:

0.8535494134022669

For our model, the Area Under the Receiver Operating Characteristic Curve (ROC AUC) score is 85%. This implies that the classification model is good enough to detect the diabetic patient.

True Value vs Predicted Value:



**5.Conclusion**

We built a machine learning-based classifier that predicts if a patient is diabetic or not, based on the information provided in the database.

While building this predictor, we learned about common preprocessing steps such as feature scaling and imputing missing values.

We implemented Random forest algorithm, evaluated the performance using the accuracy score, comparing the performance between train and test data. You can also tune the parameters and try improving the accuracy score, AUC.