

AI BASED DIABETES PREDICTION SYSTEM

Project 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:

1.Data Collection:

For data collection, you might consider collaborating with healthcare institutions, obtaining ethical approval, and ensuring strict adherence to privacy regulations. Alternatively, explore publicly available datasets from reliable sources like government health agencies or research institutions. Always prioritize privacy and ethical considerations in handling medical data.

2.Data Processing:

Data preprocessing is crucial. Start by handling missing values, normalizing numerical features, and encoding categorical variables. Clean outliers, address imbalances in the target variable (diabetes status), and split the dataset into training and testing sets. Ensure the preprocessing steps align with the requirements of the machine learning algorithms you plan to use.

3.Feature Selection:

Use techniques like correlation analysis, feature importance from treebased models, or dimensionality reduction methods such as Principal Component Analysis (PCA) to identify and select relevant features that significantly impact diabetes risk prediction. This step helps improve model efficiency and interpretability.

4.Model Selection:

Indeed, experimenting with different algorithms is a good approach. Logistic Regression is suitable for binary classification like diabetes prediction. Random Forest and Gradient Boosting are powerful ensemble methods that can capture complex relationships in the data. Evaluate their performance using metrics like

accuracy, precision, recall, and F1 score to choose the most effective model for your specific use case.

5.Evaluation:

Excellent choices for model evaluation! Accuracy provides an overall measure, precision and recall give insights into false positives and false negatives, F1-score balances precision and recall, and ROC-AUC assesses the model's ability to distinguish between classes. Consider the specific goals of your diabetes prediction task when interpreting these metrics.

6.Iterative Improvement:

Iterative improvement is a sound strategy. Utilize techniques like grid search or random search for hyperparameter tuning. Additionally, consider refining features or incorporating domain-specific knowledge for better model performance. Regularly reassess and fine-tune to enhance the accuracy and robustness of your diabetes prediction model.

Problem Statement :

The main objective is to develop an AI Based Diabetes Prediction System to predict whether a patient has diabetes or not, based on the diagnostic measurements gathered in the database using Machine learning algorithms.

Algorithm used:

Out of the machine learning algorithms used, Random tree algorithm is chosen since it has better accuracy and the following steps are taken;

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.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

importing dataset df = pd.read_csv('../input/pima-indians-diabetes-database/diabetes.csv')
df.head()
```

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.

```
df.dtypes
df.info()
```

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.

```
sns.heatmap(df.corr())
```

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

```
pair plot(df,hue='Outcome')
```

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']
```

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. 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.

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 “**rcl**”. We evaluate the performance of our model using test dataset.

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. For a detailed evaluation of our model, we look at the confusion matrix.

ROC curve

We will plot a ROC curve to determine the working of our model, If the Area Under the Receiver Operating Characteristic Curve (**ROC AUC**) score is high . This implies that the classification model is good enough to detect the diabetic patient.

Development Part 1:

In this part you will begin building your project by loading and preprocessing the dataset. Start building the AI based diabetes prediction model by loading and preprocessing the dataset. Load the diabetes prediction dataset and preprocess the data.

Data Source:

A good data source for diabetes prediction using machine learning should be Accurate whether the person has diabetes or not.

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

Pregnancy	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0
5	117	92	0	0	34.1	0.337	38	0
5	109	75	26	0	36	0.546	60	0
3	158	76	36	245	31.6	0.851	28	1
3	88	58	11	54	24.8	0.267	22	0
6	92	92	0	0	19.9	0.188	28	0
10	122	78	31	0	27.6	0.512	45	0

Import Libraries:

Import the necessary libraries for your project. You'll likely need libraries such as Pandas and NumPy and seaborn.

[Source Code:](#)

```
import pandas as pd import
```

```
numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the dataset:

Load your diabetes prediction dataset into a Pandas Data Frame. You can typically load data from a CSV file using the `pd.read_csv()` function .

Source Code: `df = pd.read_csv('../input/pima-indians-diabetes-`

`database/diabetes.csv')`

`df.head()`

Explore the data:

Take a look at the data to understand its structure and content. You can use functions like `data.head()`, `data.info()`, and `data.describe()` to get an overview.

Source Code:

```
# Display the first few rows of the dataset print(data.head())
# Get information about the dataset print(data.info())
# Summary statistics of the data print(data.describe())
```

Data Preprocessing:

- **Handling Missing Data:**

Check for missing values in the dataset and decide how to handle them (e.g., by filling missing values with the mean, median, or using other techniques).

[Source Code:](#)

```
# Handle missing values (if any)
data.fillna(method='ffill', inplace=True) # Example: Forward fill
missing values
```

- **Categorical Data:**

If your dataset contains categorical variables, you may need to encode them using techniques like one-hot encoding.

[Source Code:](#)

```
# Example one-hot encoding
data = pd.get_dummies(data, columns=['categorical_column'])
```

- **Scaling:**

Normalize or standardize numerical features to ensure that they are on a similar scale.

[Source Code:](#)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data[['numerical_column1', 'numerical_column2']] =
scaler.fit_transform(data[['numerical_column1', 'numerical_column2']])
```

Overview :

1.Data gathering and analysis:

The dataset is originally collected and circulated by “National Institute of Diabetes and Digestive and Kidney Diseases” which is available at Kaggle in the name of Pima Indians Diabetes Database. The main objective is to predict whether a patient has diabetes or not, based on the diagnostic measurements gathered in the database. All patients belong to the Pima Indian heritage, and are females of ages 21 and above.

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-diabetesdatabase/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 nonnull 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.s hape[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.s hape[0]: .2f}")
```

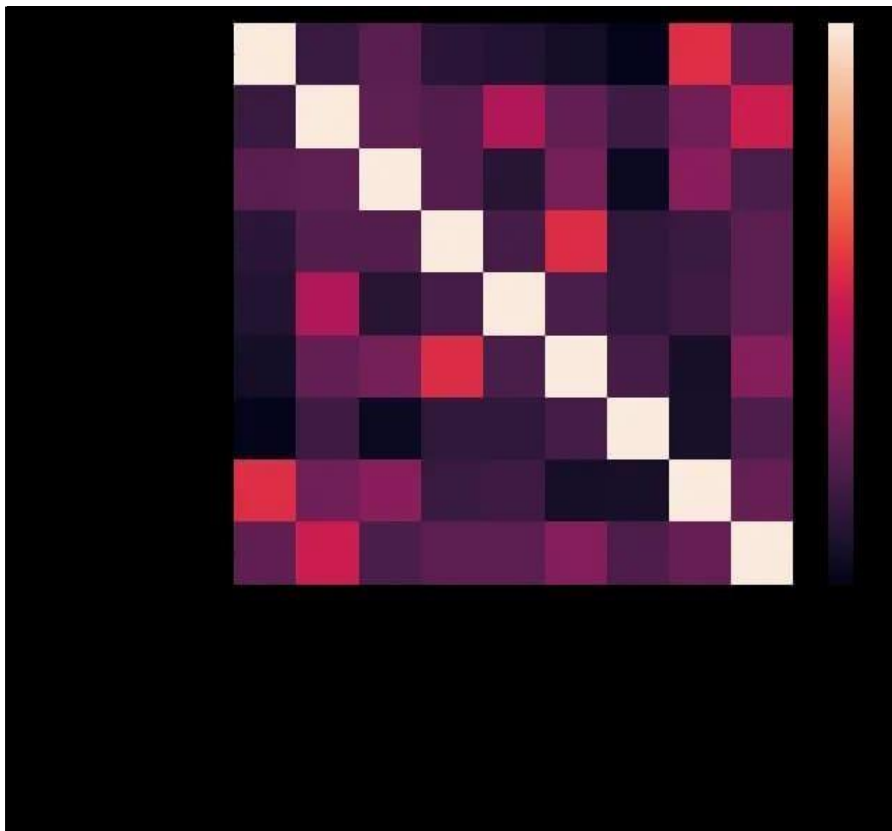
Out[8]:

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			

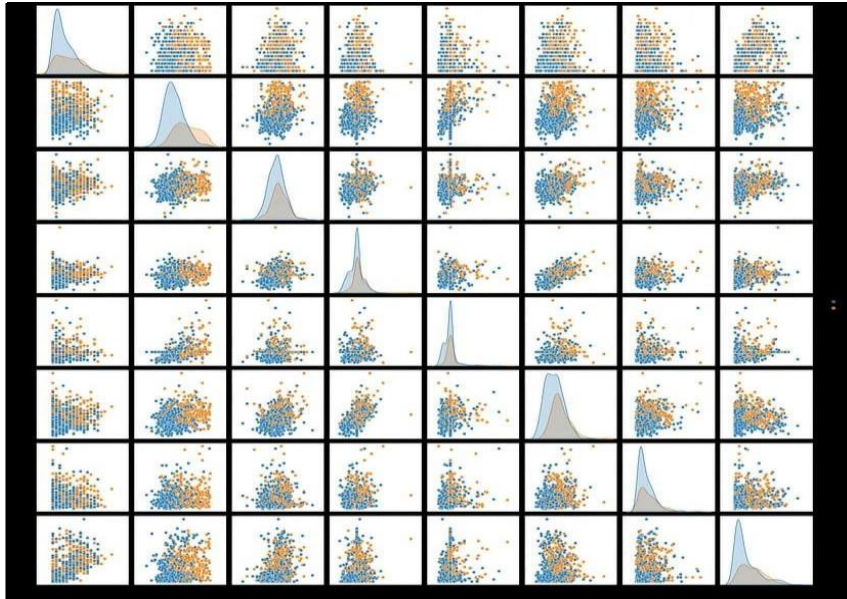
2.Data Visualization

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')
```

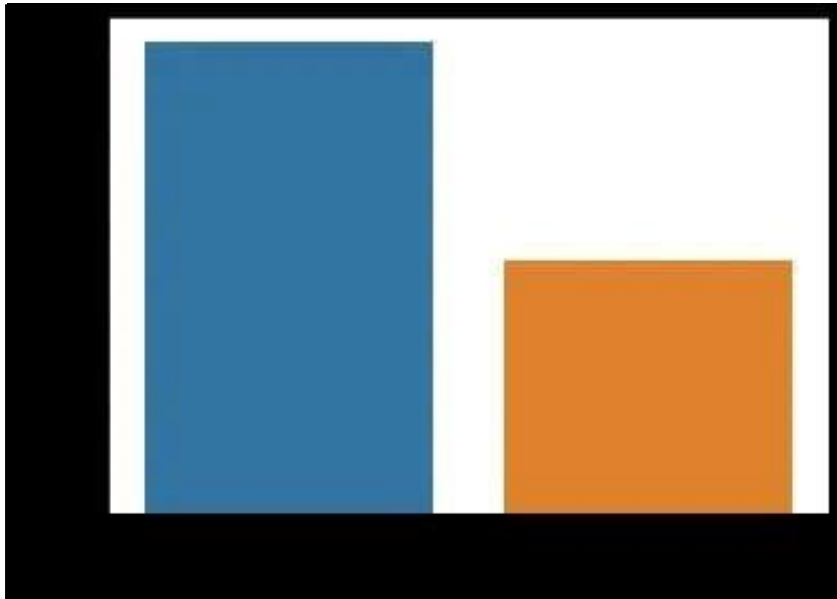


3.Classification

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.

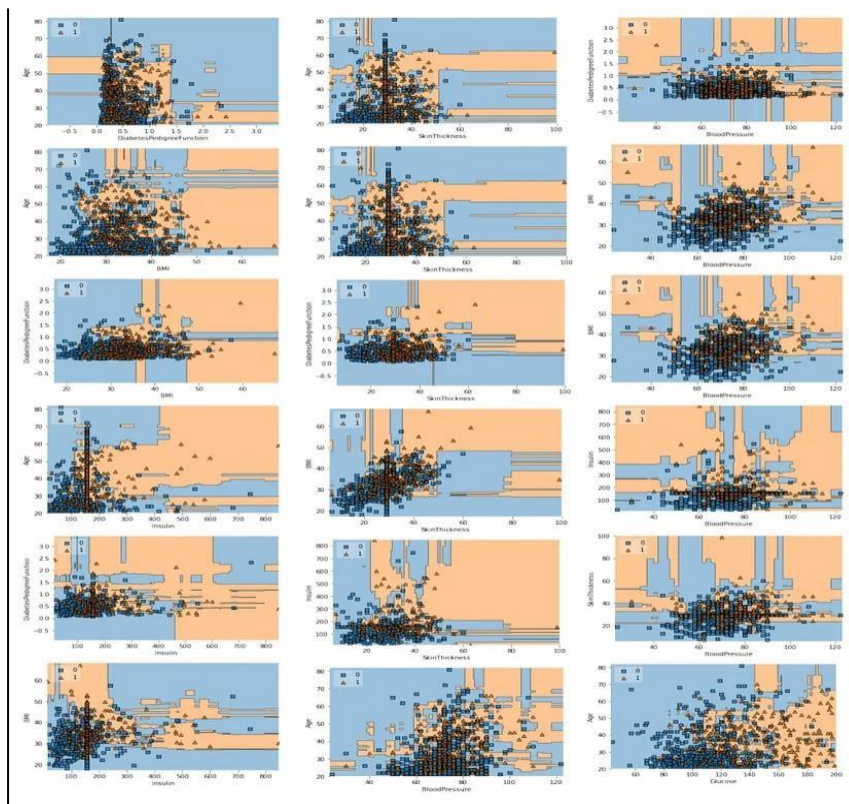
```
from mlxtend.plotting import plot_decision_regions def  
classify_with_rfc(X,Y):  
    x = df[[X,Y]].values    y =  
df['Outcome'].astype(int).values  
    rfc = RandomForestClassifier()  
    rfc.fit(x,y)  
    # Plotting decision region  
    plot_decision_regions(x, y, clf=rfc, legend=2)  
    # 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

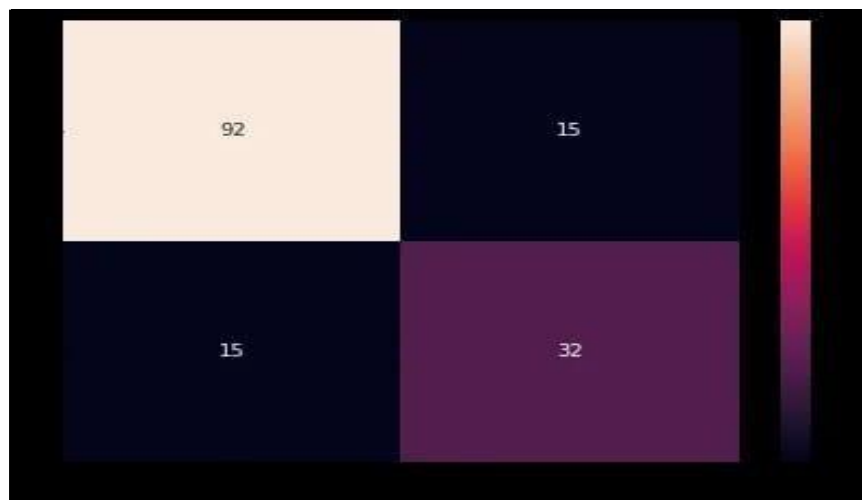


The distributions shows our model classifies the patients really well. For a detailed evaluation of our model, we look at the confusion matrix.

```

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

```

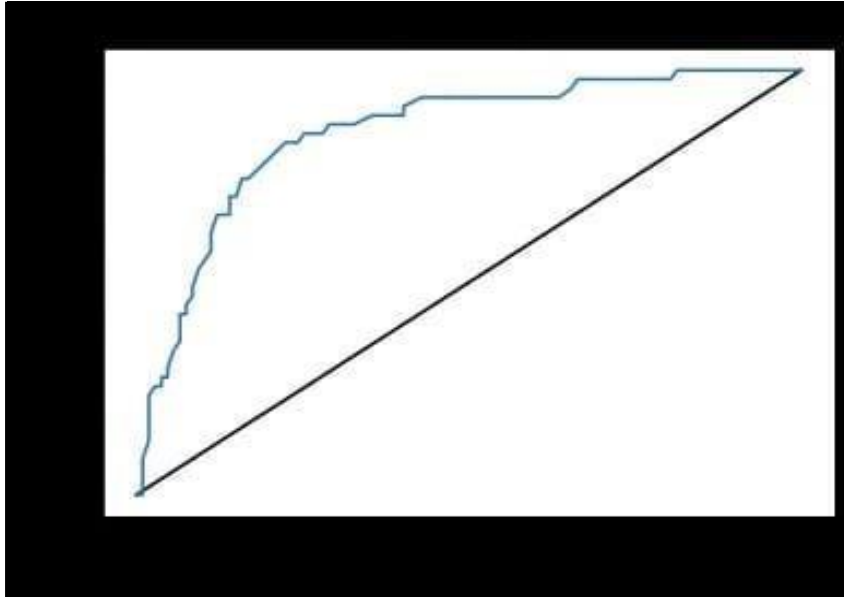
6.ROC curve

```

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()

```

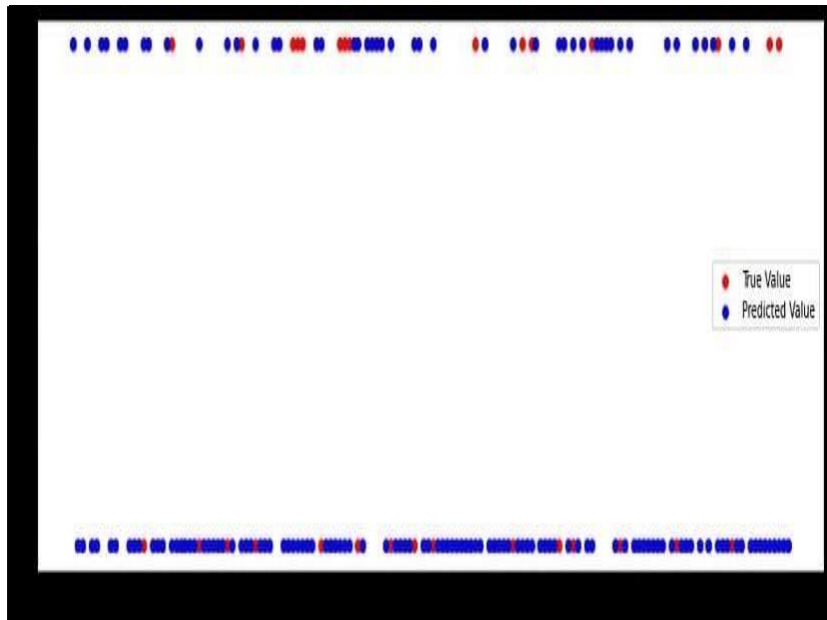
Output:



```
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.