AI_BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING.

TEAM MEMBER

922321106008: S.Deviga

Phase 5: Submission document

Project documentation and submission

Topic: In this section we will document the complete project and prepare it for submission.

AI_BASED DIABETES PREDICTION SYSTEM USING MACHINE LEARNING.

Introduction: In this phase we begin developing the diabetes prediction system by preparing the data and selecting relevant features. Loading and preprocessing data are crucial steps in building an AI-based diabetes prediction system. Proper data handling sets the foundation for developing an accurate and effective predictive model. This phase provides an overview of the significance and process of loading and preprocessing data for such a system.

Followed by these steps feature engineering, model training, and evaluation process will be continued to get AI_based diabetes prediction system model.

Given Dataset:

Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	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

Excel Dataset link:
https://in.docworkspace.com/d/sIETqyObiAYe6s6kG

769 rows *9 columns

Necessary steps to follow:

To create an AI-based diabetes prediction system, you'll need to load and work with a dataset that contains relevant information. Here are the key steps and considerations for loading a dataset:

1. **Data Collection**: Gather a dataset that includes historical information about individuals, particularly features that are relevant to diabetes prediction. Common features might include age, gender, body mass index (BMI), blood pressure, glucose levels, family history, and more.

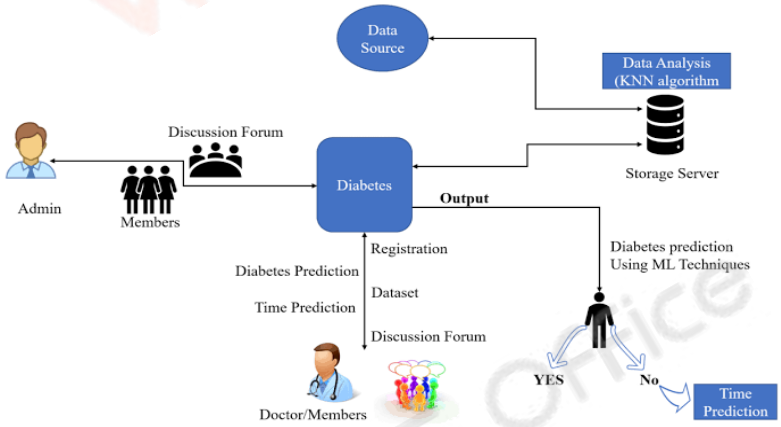
2. **Data Preprocessing**: Clean the dataset to handle missing values, outliers, and inconsistencies. This may involve imputing missing values, scaling features, and encoding categorical variables.

3. **Dataset Splitting**: Split the dataset into training, validation, and test sets. This helps in training your AI model, tuning hyperparameters, and evaluating its performance.

4.Feature Selection: Identify the most relevant features for diabetes prediction. Feature selection techniques can help in reducing dimensionality and improving model efficiency.

5.Data Normalization: Normalize or standardize the data to ensure that different features are on a similar scale. This can improve the performance of many machine learning algorithms.

6.Loading into Your AI System: Depending on the programming language and libraries you're using; you can load the dataset into your AI system. Popular libraries for this purpose include NumPy, Pandas, and scikit-learn in Python.



- Steps to be continued after loading and preprocessing the dataset.

7.Building the Model: Train your AI model, which can be a machine learning model (e.g., logistic regression, decision tree, random forest) or a deep learning model (e.g., neural network). Your model will use the loaded dataset to learn patterns and make predictions.

8.Evaluation: Assess the performance of your model using appropriate metrics like accuracy, precision, recall, F1-score, or area under the ROC curve (AUC). Tweak your model and dataset as needed for better results.

9.Deployment: Once your AI-based diabetes prediction system performs well, you can deploy it in a real-world environment where it can make predictions for new data.

10.Monitoring and Maintenance: Continuously monitor the system's performance and update the dataset and model as new data becomes available.

Loading the Dataset:

Data Collection: Obtain a dataset that contains relevant information for diabetes prediction. You can find such datasets from sources like the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK) or the UCI Machine Learning Repository.

Choose a Programming Language and Libraries: Select a programming language (e.g., Python) and

libraries (e.g., Pandas) to work with your dataset. Python is commonly used for data science and machine learning tasks.

Load the Dataset: Use library functions to load your dataset. For example, in Python, you can use Pandas to read data from CSV, Excel, or other file formats.

Data Preprocessing:

Handling Missing Values: Check for and handle missing values. You can either remove rows with missing values or impute them using techniques like mean, median, or regression imputation.

Dealing with Outliers: Identify and handle outliers in your dataset. You can use statistical methods or visualization techniques to detect outliers and then decide whether to remove or transform them.

Feature Engineering: Create new features or transform existing ones to make them more suitable for your model. For example, you can compute the body mass index (BMI) if it's not already in the dataset.

Data Scaling: Normalize or standardize numerical features to ensure they are on the same scale. This is important, especially if you plan to use algorithms

sensitive to feature scaling, like support vector machines or k-nearest neighbors.

Encoding Categorical Variables: If your dataset contains categorical variables, encode them into numerical values. One-hot encoding is a common technique for this purpose.

Split the Data: Divide your dataset into training, validation, and test sets. This is important for model training, hyperparameter tuning, and evaluation.

- With the dataset loaded and preprocessed, you can proceed to build and train your diabetes prediction model using machine learning or deep learning techniques.

PROGRAM:

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

IMPORTING SEABORN LIBRARY FOR STATISTICAL DATA VISUALIZATION

import seaborn as sns

IMPORTING MATPLOTLIB LIBRARY FOR CREATING PLOTS AND VISUALIZATIONS
import matplotlib.pyplot as plt

IMPORTING PLOTLY EXPRESS LIBRARY FOR INTERACTIVE VISUALIZATIONS

import plotly.express as px
```

Exploratory Data Analysis (EDA): It is a critical step in understanding and gaining insights from your dataset in an AI-based diabetes prediction system.

LOAD AND PREPARE DATA

```
df=pd.read_excel('https://in.docworkspace.com/d/sIETqyObiAYe6s6kG')
```

UNDERSTANDING THE VARIABLES

INPUT

```
df.head(10)
```

OUTPUT

S:NO	pregnancies	Glucose	Blood pressure	Skin thickness	insulin	BMI	Diabetes pedigree function	Age	Out come
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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

INPUT

```
df.tail(10)
```

OUTPUT

S:NO	pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes pedigree function	Age	Out come
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0

766	1	126	60	0	0	30.1	0.349	47	1

INPUT

```
df.sample(5)
```

OUTPUT

S:NO	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes pedigree function	Age	**Out come**
345	8	126	88	36	108	38.5	0.349	49	0
578	10	133	68	0	0	27.0	0.245	36	0
84	5	137	108	0	0	48.8	0.227	37	1
217	6	125	68	30	120	30.0	0.464	32	0
595	0	188	82	14	185	32.0	0.682	22	1

INPUT

```
df.describe()
```

OUTPUT

S:NO	Pregnancies	Glucose	Blood pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	**Out come**
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

INPUT

```
df.size
```

OUTPUT

6912

INPUT

```
df.shape
```

OUTPUT

```
(768, 9)
```

Data cleaning: It can be a complex and domain-specific process. The specific steps and techniques you use may vary depending on your dataset and the nature of the data quality issues.

INPUT

```
df=df.drop_duplicates()
```

```
Df.shape
```

OUTPUT

```
(768, 9)
```

CHECK FOR NULL VALUES

INPUT

```
df.isnull().sum()
```

OUTPUT

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

THERE IS NO MISSING VALUES PRESENT IN THE DATA

INPUT

```
df.columns
```

OUTPUT

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Check the number of Zero Values in Dataset

INPUT

```
print("No. of Zero Values in Glucose ", df[df['Glucose']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Glucose  5
```

INPUT

```
print("No. of Zero Values in Blood Pressure ",
df[df['BloodPressure']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Blood Pressure  35
```

INPUT

```
print("No. of Zero Values in SkinThickness ",
df[df['SkinThickness']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in SkinThickness  227
```

INPUT

```
print("No. of Zero Values in Insulin ", df[df['Insulin']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in Insulin  374
```

INPUT

```
print("No. of Zero Values in BMI ", df[df['BMI']==0].shape[0])
```

OUTPUT

```
No. of Zero Values in BMI  11
```

Replace zeroes with mean of that Columns

INPUT

```
df['Glucose']=df['Glucose'].replace(0, df['Glucose'].mean())
print('No of zero Values in Glucose ', df[df['Glucose']==0].shape[0])

No of zero Values in Glucose  0


df['BloodPressure']=df['BloodPressure'].replace(0,
df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0,
df['SkinThickness'].mean())
df['Insulin']=df['Insulin'].replace(0, df['Insulin'].mean())
df['BMI']=df['BMI'].replace(0, df['BMI'].mean())

Validate the Zero Values:

df.describe()
```

OUTPUT

	Pregnancies	glucose	Blood Pressure	Skin Thickness	Insulin	Insulin	Diabetes Pedigree Function	Age	Out come
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Data visualization: Data visualization helps you understand the data distribution, relationships between features, and can be particularly useful for feature selection and feature engineering. You can use libraries like Matplotlib, Seaborn, or Plotly in Python to create these visualizations.

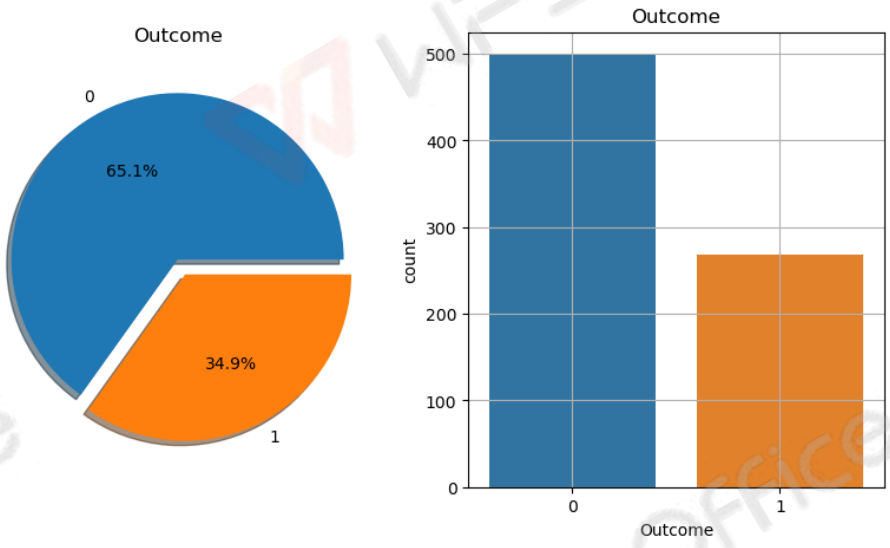
Bar Charts: Use bar charts to display the distribution of categorical features such as gender, family history, and medication usage.

INPUT

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
f, ax = plt.subplots(1, 2, figsize=(10, 5))
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%',
ax=ax[0], shadow=True)
ax[0].set_title('Outcome')
ax[0].set_ylabel(' ')
sns.countplot(x='Outcome', data=df, ax=ax[1])  # Use 'x' instead of
'Outcome'
ax[1].set_title('Outcome')
N, P = df['Outcome'].value_counts()
print('Negative (0):', N)
print('Positive (1):', P)
plt.grid()
plt.show()
```

OUTPUT

```
Negative (0): 500
Positive (1): 268
```



1 Represent --> Diabetes Positive
0 Represent --> Diabetes Negative

Histograms: Visualize the distribution of numerical features like age, glucose levels, BMI, etc. Histograms help you understand the data's central tendencies and spread.

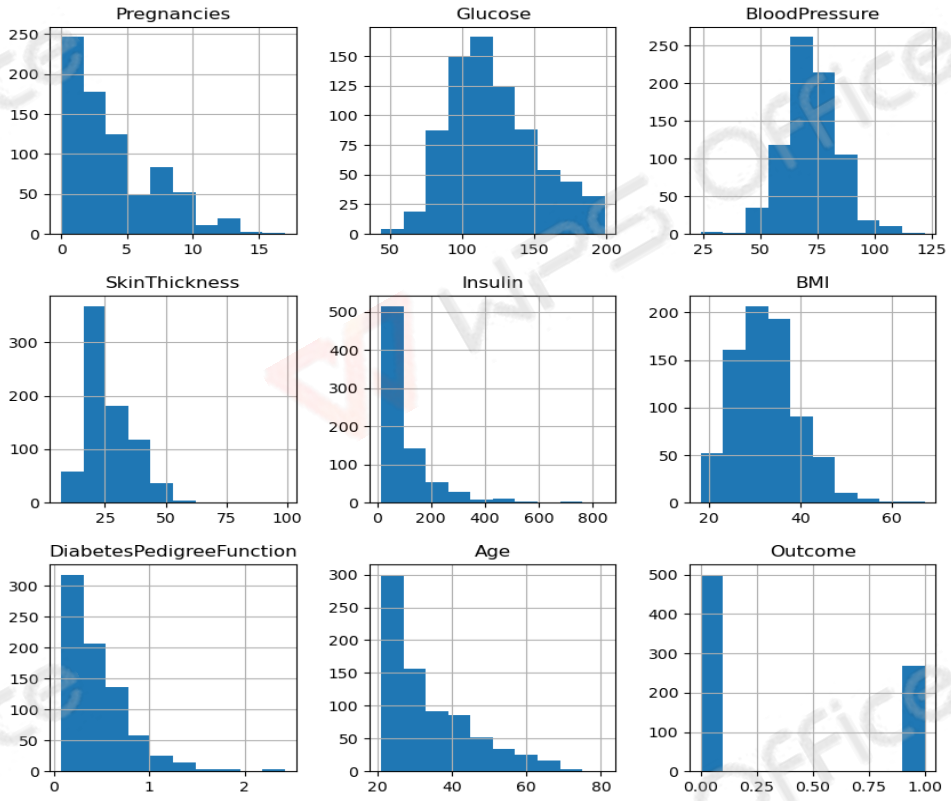
Pie Charts: Use pie charts to visualize the distribution of categorical variables, such as the percentage of people with and without diabetes.

Line Charts: If your data has a temporal aspect, create line charts to observe trends over time. For example, tracking glucose levels over weeks

INPUT

```
df.hist(bins=10, figsize=(10, 10))
plt.show()
```

OUTPUT

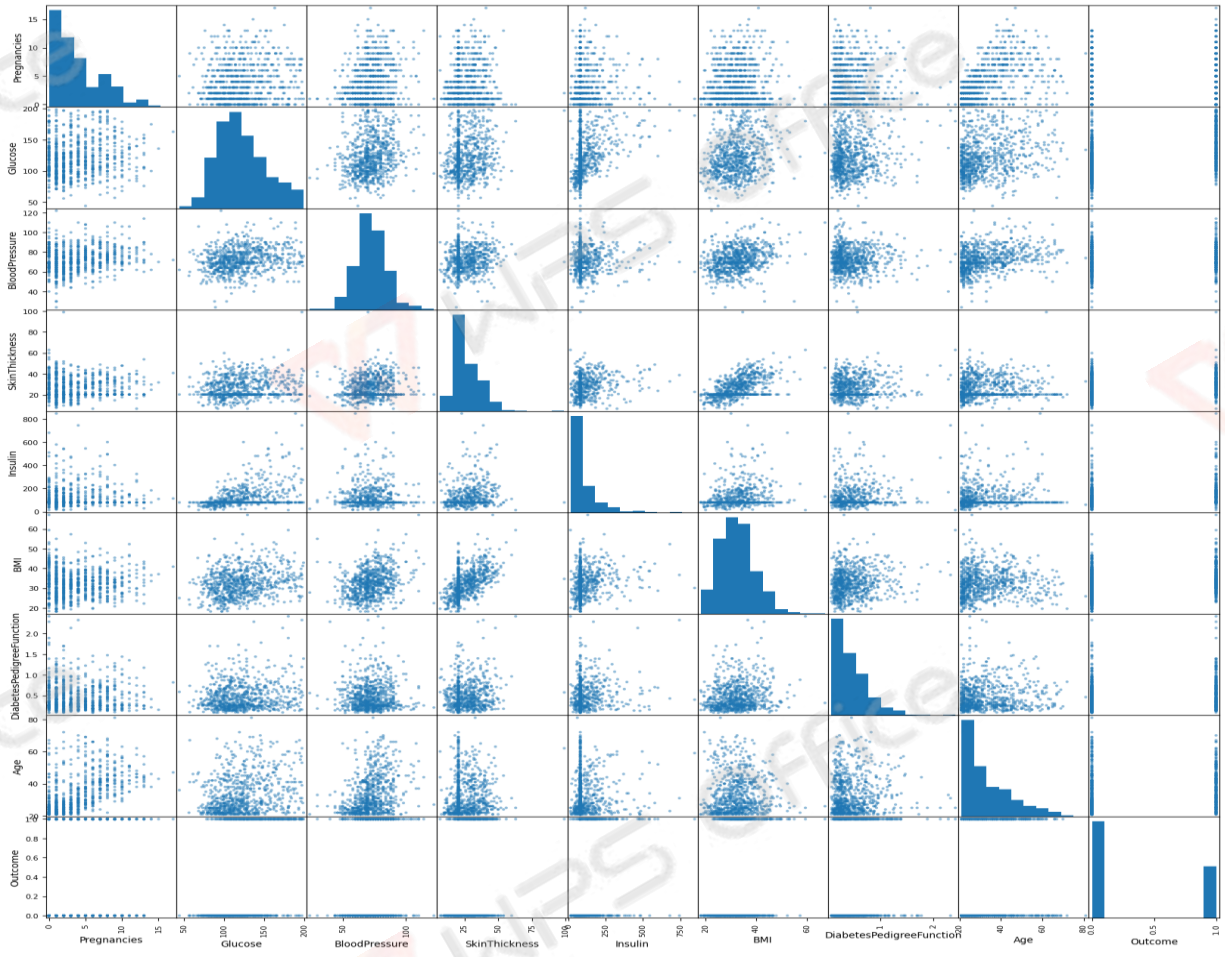


Scatter Plots: Plot relationships between two numerical features to identify correlations and patterns. For instance, you can plot glucose levels against BMI .

INPUT

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize =(20, 20))
```

OUTPUT



Pair Plots: Plot pairs of features against each other, especially when you have multiple numerical features. Pair plots help in visualizing relationships within the dataset.

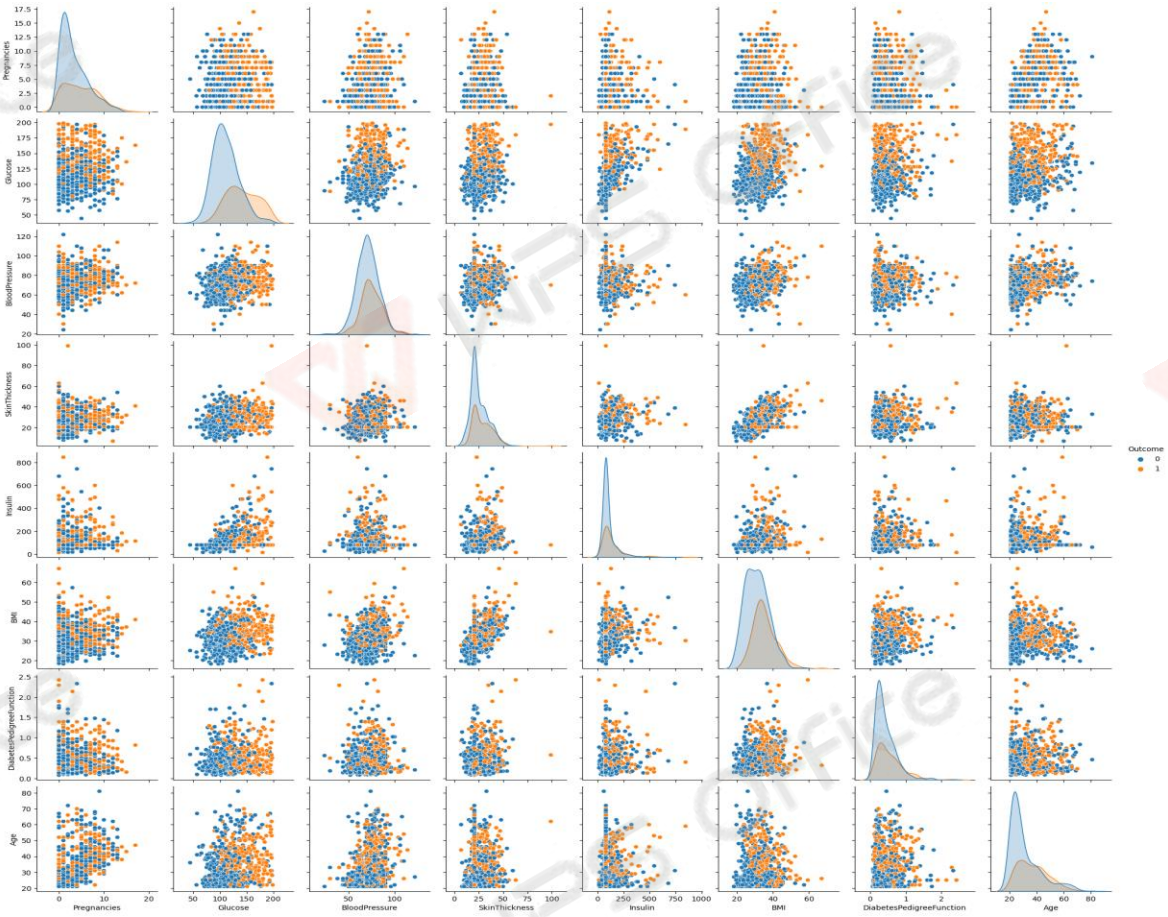
Pair Plots: Plot pairs of features against each other, especially when you have multiple numerical features. Pair plots help in visualizing relationships within the dataset.

Violin Plots: Violin plots combine a box plot and a kernel density plot to visualize the distribution of numerical features.

INPUT

```
sns.pairplot(data=df, hue='Outcome')
plt.show()
```

OUTPUT:

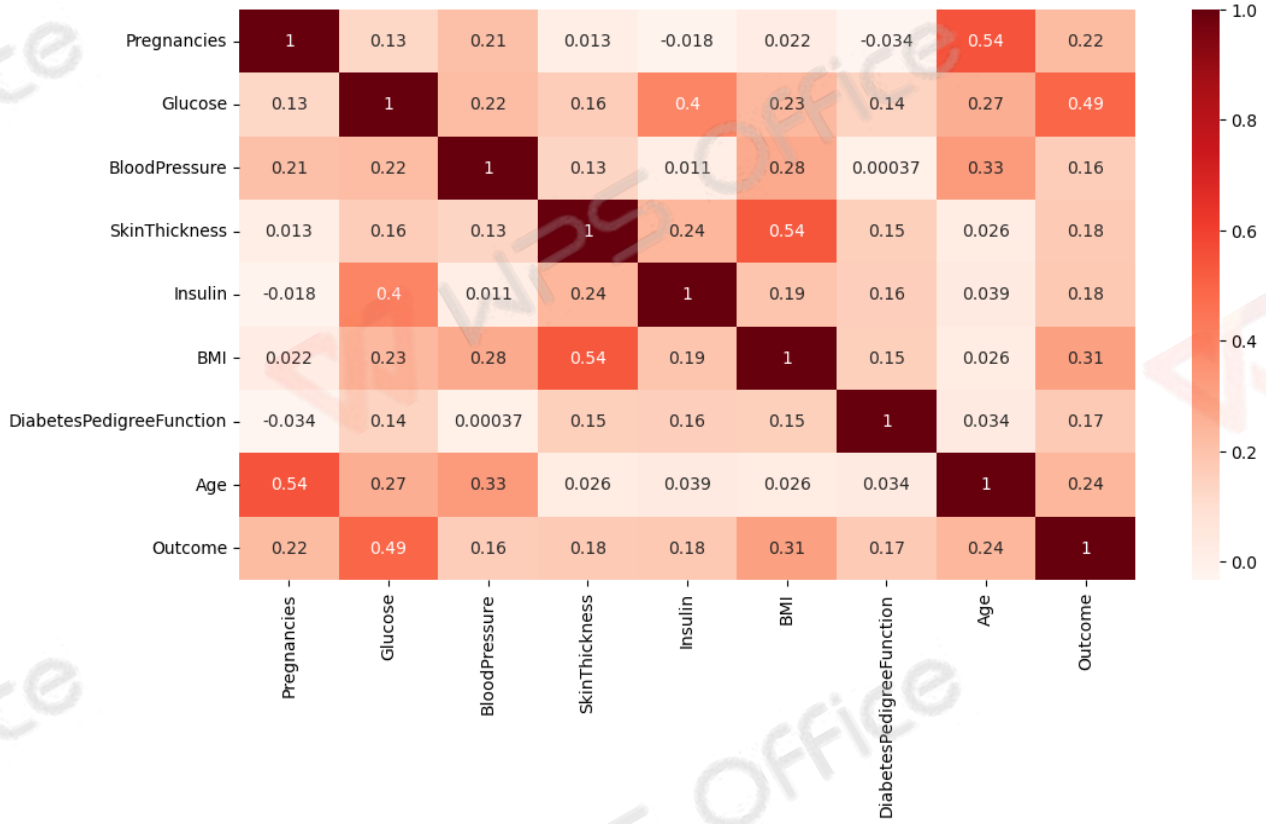


Correlation Heatmaps: Create a heatmap to show the correlation between numerical features. This helps you understand how features relate to each other.

INPUT

```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(), annot=True, cmap='Reds')
plt.plot()
```

OUTPUT



INPUT

```
mean = df['Outcome'].mean()
```

OUTPUT

```
0.3489583333333333
```

SPLIT THE DATA FRAME INTO X AND Y

INPUT

```
target_name='Outcome'
y=df[target_name]
X= df.drop(target_name, axis=1)

X.head()
```

OUTPUT

S:NO	Pregnancies	glucose	Blood pressure	Skin thickness	Insulin	BMI	Diabetes Pedigree function	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

INPUT

```
y.head()
```

OUTPUT

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

FUTURE SCALING: Scaling an AI-based diabetes prediction system involves a multidisciplinary approach that combines expertise in healthcare, AI, data science, and technology. Continuous learning, adaptation, and collaboration are key elements in the successful scaling of such system

INPUT

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2,
random_state=7)
```

```
X_train.shape, y_train.shape
```

OUTPUT

```
((614, 8), (614,))
```

INPUT

```
X_test.shape, y_test.shape
```

OUTPUT

```
((154, 8), (154,))
```

- The following are the steps to be followed after completing the loading and preprocessing of dataset.

FEATURE ENGINEERING:

Feature engineering is the process of selecting, transforming, and creating relevant features (input variables) from the available data that will be used to train your AI model. In the context of a diabetes prediction system, some common features might include age, BMI, family medical history, glucose levels, and more. Feature engineering also involves handling missing data, scaling, and encoding categorical variables, making the data suitable for machine learning.

MODEL TRAINING:

Model training involves the development of a machine learning or deep learning model using your dataset. In the case of diabetes prediction, you might use algorithms like logistic regression, decision trees, random forests, support

vector machines, or neural networks. The model learns patterns from the features in your dataset and makes predictions.

EVALUATION:

After training your model, you need to assess its performance. Common evaluation metrics for a binary classification problem like diabetes prediction include accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC). You'll typically split your dataset into a training set and a test set to evaluate how well your model generalizes to new, unseen data. Cross-validation can also be used to assess the model's robustness.

It's important to note that the choice of features, the model architecture, and the evaluation metrics may vary depending on the specific requirements and characteristics of your dataset and application. Continuous refinement and fine-tuning of these components are often necessary to build an accurate and reliable diabetes prediction system.

Here is the further procedure to proceed after data collection and data processing process.

Model selection: It is an AI-based diabetes prediction system that involves choosing the most

appropriate machine learning or statistical model to achieve accurate predictions. Here's a simplified process:

Data Collection: Gather a dataset containing features (e.g., age, BMI, glucose levels) and target labels (diabetes status).

Data Preprocessing: Clean and preprocess the data by handling missing values, scaling features, and encoding categorical variables.

Split Data: Divide the dataset into training, validation, and test sets to evaluate model performance properly.

Model Selection: Consider various machine learning models such as logistic regression, decision trees, random forests, support vector machines, neural networks, and more. Factors to consider:

- Complexity of the model.
- Interpretability of the model.
- Performance metrics (accuracy, precision, recall, F1-score) on the validation set.
- Computational resources required.

Hyperparameter Tuning: For selected models, fine-tune hyperparameters (e.g., learning rate, depth of

trees, number of hidden layers) using techniques like grid search or random search.

CROSS-VALIADATION: Perform k-fold cross-validation to ensure the model's generalization ability and reduce overfitting.

EVALUATE PERFORMANCE: Assess models on the test set using appropriate evaluation metrics for diabetes prediction. Consider the trade-offs between false positives and false negatives, as it depends on the application's requirements.

ENSEMBLE METHODS: Consider using ensemble techniques like bagging (e.g., Random Forests) or boosting (e.g., AdaBoost, Gradient Boosting) to combine multiple models for improved performance.

MODEL INTERPRETABILITY: Depending on the application, choose models that provide interpretability, such as decision trees or linear regression, to understand the factors influencing predictions.

DEPLOYMENT: Once you've selected the best-performing model, deploy it in your diabetes prediction system for real-world use.

PROGRAM:

INPUT:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(SSX, y, test_size=0.2,
random_state=7)


X_train.shape, y_train.shape
```

OUTPUT:

```
((614, 8), (614,))
```

INPUT

```
X_test.shape, y_test.shape
```

OUTPUT

```
((154, 8), (154,))
```

Logistic Regression: Logistic regression can be a valuable tool in an AI-based diabetes prediction system

- Use logistic regression to build a predictive model. Logistic regression is well-suited for binary classification task Assess the performance of your logistic regression model using metrics like accuracy, precision, recall, F1-score, and ROC AUC. Cross-validation can help estimate model robustness
- Consider using L1 or L2 regularization to prevent overfitting and improve model generalization.

- Logistic regression models are inherently interpretable, making it easier to explain to stakeholders and healthcare professionals why a certain prediction was made
- **CLASSFICATION ALGORITHMS:**

LOGISTIC REGRESSION:

INPUT

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

OUTPUT

```
LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

- Decision Tree: use a decision tree algorithm to build a predictive model. Decision trees are useful for both classification and regression tasks. In this case, you would use it for binary classification (diabetes or no diabetes).
- Decision trees can automatically rank feature importance, helping you understand which factors contribute the most to diabetes prediction.
- Assess the performance of your decision tree model using metrics like accuracy, precision, recall, F1-score, and ROC AUC. Cross-validation can help estimate model robustness.

- Decision trees can be prone to overfitting, so consider pruning the tree to improve generalization and avoid making overly complex decisions. Combine multiple decision trees using ensemble methods like Random Forests or Gradient Boosting to enhance prediction accuracy and reduce overfitting.
- Decision trees are inherently interpretable, making it easy to explain the reasoning behind a prediction to healthcare professionals and patients.

MAKING PREDICTION:

Logistic Regression:

INPUT

```
X_test.shape
```

OUTPUT

(154, 8)

INPUT

```
lr_pred=lr.predict(X_test)
lr_pred.shape
```

OUTPUT

```
(154,)
```

Decision Tree:

```
dt_pred=dt.predict(X_test)
dt_pred.shape
```

OUTPUT

```
(154,)
```

Accuracy: Accuracy is a fundamental performance metric in an AI-based diabetes prediction system. It

provides an overall measure of how well the model is performing. Here's how accuracy is relevant to such a system:

Accuracy is calculated as:

Accuracy= correct predictions / total predictions

Correct Predictions: The sum of true positives (correctly predicted cases of diabetes) and true negatives (correctly predicted cases of non-diabetes).

Total Predictions: The total number of predictions made by the model.

- A high accuracy indicates that the model is making a large proportion of correct predictions, both for patients with diabetes and those without it. It's a straightforward and easy-to-understand metric that can provide an initial assessment of the model's performance.

However, it's important to note that accuracy might not be the only relevant metric, especially in healthcare applications. Diabetes prediction systems often deal with imbalanced datasets, where non-diabetic cases significantly outnumber diabetic cases. In such cases, a model can achieve high accuracy by simply predicting "no diabetes" for every case, but this would not be clinically useful.

To get a more comprehensive evaluation, consider using additional metrics like precision, recall, and the F1-score, which provide insights into the model's ability to correctly predict diabetes cases and non-diabetes cases while accounting for imbalances in the dataset

MODEL EVALUATION FOR LOGISTIC REGRESSION:

INPUT

```
from sklearn.metrics import accuracy_score
print("Train Accuracy of Logistic Regression: ", lr.score(X_train,
y_train)*100)
print("Accuracy (Test) Score of Logistic Regression: ", lr.score(X_test,
y_test)*100)
print("Accuracy Score of Logistic Regression: ", accuracy_score(y_test,
lr_pred)*100)
```

OUTPUT

```
Train Accuracy of Logistic Regression:  77.36156351791531
Accuracy (Test) Score of Logistic Regression:  77.27272727272727
Accuracy Score of Logistic Regression:  77.27272727272727
```

MODEL EVALUATION FOR DECISION TREE

INPUT

```
print ("Train Accuracy of Decesion Tree: ", dt.score(X_train,
y_train)*100)
print("Accuracy (Test) Score of Decesion Tree: ", dt.score(X_test,
y_test)*100)
print("Accuracy Score of Decesion Tree: ", accuracy_score(y_test,
dt_pred)*100)
```

OUTPUT

```
Train Accuracy of Decesion Tree:  100.0
Accuracy (Test) Score of Decesion Tree:  81.16883116883116
Accuracy Score of Decesion Tree:  81.16883116883116
```

Confusion matrix: A confusion matrix is a valuable tool in assessing the performance of an AI-based diabetes prediction system. It provides a detailed breakdown of the model's predictions, allowing you to understand how well it's classifying diabetes and non-diabetes cases. The confusion matrix is especially useful in the context of binary classification, where you have two classes: diabetes (positive) and non-diabetes (negative).

A confusion matrix consists of four key components:

True Positives (TP): The number of cases correctly predicted as having diabetes .

True Negatives (TN): The number of cases correctly predicted as not having diabetes.

False Positives (FP): The number of cases incorrectly predicted as having diabetes when they do not (Type I error).

False Negatives (FN): The number of cases incorrectly predicted as not having diabetes when they do have diabetes (Type II error).

With this information, you can calculate various performance metrics, such as:

Accuracy: TP+TN/TP+TN+FP+FN

Precision:TP/TP+FP

Recall(sensitivity):TP/TP+FN

Specificity: TN/TN+FP

F1-Score: A measure that balances precision and recall, computed as

2.precision. recall/precision+recall

The confusion matrix and these associated metrics provide a comprehensive view of your model's performance, helping you understand its ability to correctly predict diabetes cases while minimizing false alarms or missed cases. This information is crucial for assessing the clinical utility of your diabetes prediction system.

INPUT

```
from sklearn.metrics import classification_report, confusion_matrix
cm = confusion_matrix(y_test, lr_pred)
cm
```

OUTPUT

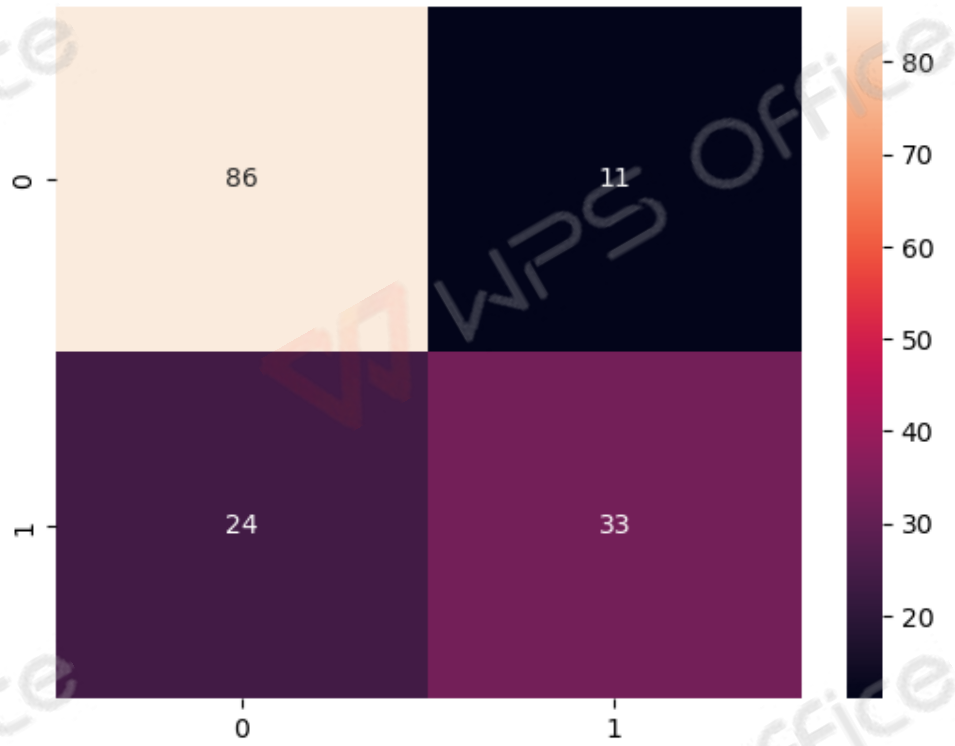
```
array ([[86, 11],
       [24, 33]])
```

INPUT

```
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")
```

OUTPUT

```
<Axes: >
```



INPUT

```
TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP  = cm[1,1]

TN, FP, FN, TP
```

OUTPUT

```
(86, 11, 24, 33)
```

INPUT

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, lr_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
```

```
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1])
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]),
np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],
cm[1,0]]), np.sum(cm))*100))
```

OUTPUT

```
TN - True Negative 86
FP - False Positive 11
FN - False Negative 24
TP - True Positive 33
Accuracy Rate: 77.27272727272727
Misclassification Rate: 22.727272727272727
```

INPUT

```
77.27272727272727+22.727272727272727
```

OUTPUT

```
100.0
```

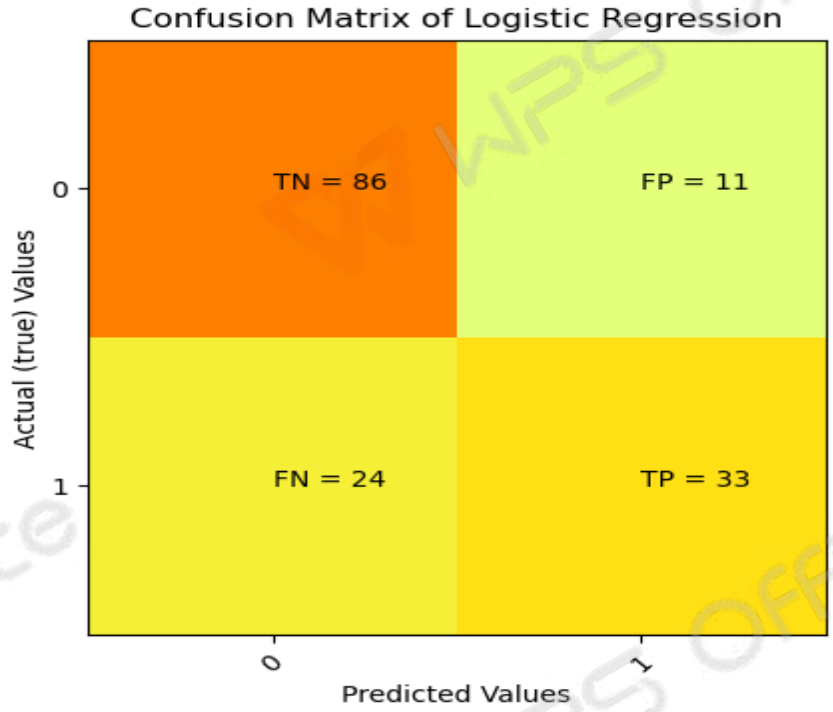
INPUT

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

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Logistic Regression')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

OUTPUT



INPUT

```
pd.crosstab(y_test, lr_pred, margins=False)
```

OUTPUT

col_0	0	1
Outcome		
0	86	11
1	24	33

INPUT

```
pd.crosstab(y_test, lr_pred, margins=True)
```

OUTPUT

col_0	0	1	All
Outcome			
0	86	11	97
1	24	33	57
All	110	44	154

INPUT

```
pd.crosstab(y_test, lr_pred, rownames=['Actual values'],
colnames=['Predicted values'], margins=True)
```

OUTPUT

Predicted values	0	1	All
Actual values			
0	86	11	97
1	24	33	57
All	110	44	154

Precision:

PPV- positive Predictive Value

Precision = True Positive/True Positive + False Positive
Precision = TP/TP+FP

INPUT

```
TP, FP
```

OUTPUT

```
(33, 11)
```

INPUT

```
Precision = TP/(TP+FP)
```

OUTPUT

```
0.75
```

INPUT

```
33/ (33+11)
```

OUTPUT

```
0.75
```

precision Score:

INPUT

```
precision_score = TP/float(TP+FP)*100
print('Precision Score: {0:0.4f}'.format(precision_score))
```

OUTPUT

```
Precision Score: 75.0000
```

INPUT

```
from sklearn.metrics import precision_score
print("Precision Score is: ", precision_score(y_test, lr_pred)*100)
print("Micro Average Precision Score is: ", precision_score(y_test,
lr_pred, average='micro')*100)
print("Macro Average Precision Score is: ", precision_score(y_test,
lr_pred, average='macro')*100)
print("Weighted Average Precision Score is: ", precision_score(y_test,
lr_pred, average='weighted')*100)
print("precision Score on Non Weighted score is: ",
precision_score(y_test, lr_pred, average=None)*100)
```

OUTPUT

```
Precision Score is:  75.0
Micro Average Precision Score is:  77.27272727272727
Macro Average Precision Score is:  76.5909090909091
```

```
Weighted Average Precision Score is:  77.00413223140497
precision Score on Non Weighted score is:  [78.18181818 75. ]
```

INPUT

```
print('Classification Report of Logistic Regression: \n',
classification_report(y_test, lr_pred, digits=4))
```

OUTPUT

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

Precision:

Precision is an important performance metric in an AI-based diabetes prediction system. It measures the accuracy of positive predictions made by the model. Here's how precision is relevant to such a system:

Precision is the ratio of true positive predictions to the total positive predictions made by the model. In the context of a diabetes prediction system:

True Positives (TP): These are cases where the model correctly predicts a patient as having diabetes, and they indeed have diabetes.

False Positives (FP): These are cases where the model incorrectly predicts a patient as having diabetes, but they do not have diabetes.

Precision is calculated as:

True positives/(true positives+ false positives)

- A high precision indicates that when the model predicts a patient has diabetes, it's more likely to be correct. In a healthcare setting, high precision is crucial because it means fewer false alarms or unnecessary treatments. It helps in avoiding unnecessary stress and expenses for patients and healthcare resources.

However, precision should be considered in conjunction with other metrics like recall and F1-score, as there's often a trade-off between precision and recall. Achieving a balance between precision and recall is essential to ensure that the model correctly identifies diabetes cases while minimizing false alarms.

```
True Positive Rate(TPR)

Recall = True Positive/True Positive + False Negative
Recall = TP/TP+FN
```

INPUT

```
recall_score = TP/ float(TP+FN)*100
print('recall_score', recall_score)
```

OUTPUT

```
recall_score 57.89473684210527
```

INPUT

TP, FN

OUTPUT

```
(33, 24)
```

INPUT

```
33/(33+24)
```

OUTPUT

```
0.5789473684210527
```

INPUT

```
from sklearn.metrics import recall_score
print('Recall or Sensitivity_Score: ', recall_score(y_test, lr_pred)*100)
```

```
Recall or Sensitivity_Score:  57.89473684210527
```

```
print("recall Score is: ", recall_score(y_test, lr_pred)*100)
print("Micro Average recall Score is: ", recall_score(y_test, lr_pred,
average='micro')*100)
print("Macro Average recall Score is: ", recall_score(y_test, lr_pred,
average='macro')*100)
print("Weighted Average recall Score is: ", recall_score(y_test, lr_pred,
average='weighted')*100)
print("recall Score on Non Weighted score is: ", recall_score(y_test,
lr_pred, average=None)*100)
```

OUTPUT

```
recall Score is:  57.89473684210527
Micro Average recall Score is:  77.27272727272727
Macro Average recall Score is:  73.27726532826912
```

Weighted Average recall Score is: 77.27272727272727
recall Score on Non Weighted score is: [88.65979381 57.89473684]

INPUT

```
print('Classification Report of Logistic Regression: \n',
classification_report(y_test, lr_pred, digits=4))
```

OUTPUT

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

FPR - False Positve Rate

INPUT

```
FPR = FP / float(FP + TN) * 100
print('False Positive Rate: {:.4f}'.format(FPR))
```

OUTPUT

```
False Positive Rate: 11.3402
```

INPUT

FP, TN

OUTPUT

```
(11, 86)
```

INPUT

```
11/(11+86)
```

OUTPUT

```
0.1134020618556701
```

Recall: Recall, also known as sensitivity or true positive rate, is an important performance metric in an AI-based diabetes prediction system. It measures the model's ability to correctly identify all positive cases, or in this context, patients with diabetes. Here's how recall is relevant to such a system:

True Positives (TP): These are cases where the model correctly predicts a patient as having diabetes, and they indeed have diabetes.

False Negatives (FN): These are cases where the model incorrectly predicts a patient as not having diabetes, but they do have diabetes.

Recall is calculated as

Recall: True positives/(True positives+false positives)

A high recall indicates that the model is effective at identifying patients with diabetes, minimizing the risk of missing true cases. In the context of a diabetes prediction system, high recall is crucial because it means the system is better at identifying individuals who need medical attention or diabetes management.

However, recall should be considered in conjunction with other metrics like precision and the F1-score. There's

often a trade-off between precision and recall. While high recall minimizes false negatives, it may result in more false positives. Achieving a balance between precision and recall is essential to ensure that the model correctly identifies diabetes cases while minimizing incorrect predictions.

INPUT

```
specificity = TN /(TN+FP)*100
print('Specificity : {0:0.4f}'.format(specificity))
```

OUTPUT

```
Specificity : 88.6598
```

INPUT

```
from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

OUTPUT

```
F1_Score of Macro:  65.34653465346535
```

INPUT

```
print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred,
average='micro')*100)
print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred,
average='macro')*100)
print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred,
average='weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred,
average=None)*100)
```

OUTPUT

```
Micro Average f1 Score is:  77.27272727272727
Macro Average f1 Score is:  74.21916104653944
Weighted Average f1 Score is:  76.52373933045479
```

```
f1 Score on Non Weighted score is:  [83.09178744 65.34653465]
```

Classification Report of Logistic Regression:

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n',
classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve& ROC AUC

INPUT

```
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

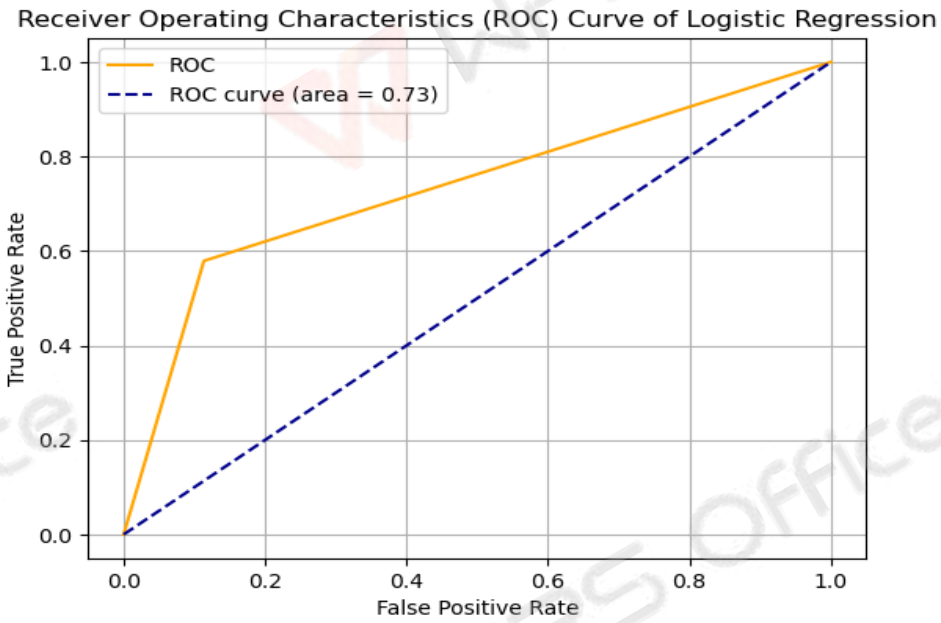
```
ROC AUC SCORE of logistic Regression is  0.7327726532826913
```

INPUT

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC
curve (area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic
Regression")
plt.legend()
plt.grid()
plt.show()
```

OUTPUT



Confusion Matrix of Decision tree:

Confusion Matrix:

```
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, dt_pred)
cm
```

OUTPUT

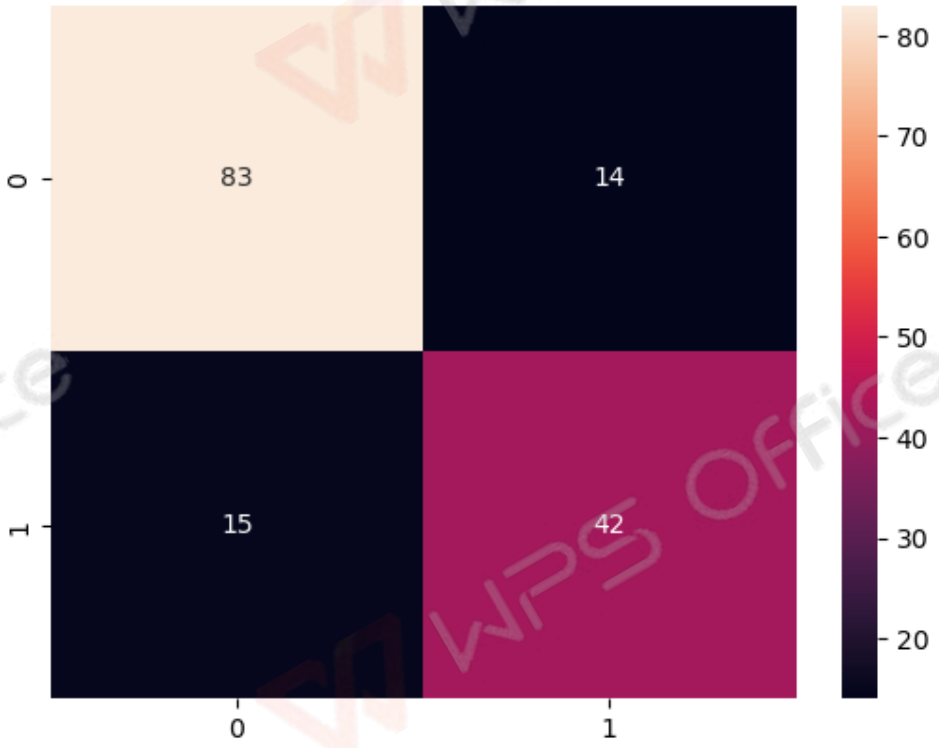
```
array([[83, 14],
       [15, 42]])
```

INPUT

```
sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

OUTPUT:

```
<Axes: >
```



INPUT

```
TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP  = cm[1,1]
```

INPUT

TN, FP, FN, TP

OUTPUT

```
(83, 14, 15, 42)
```

INPUT

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, dt_pred)

print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]),
np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],
cm[1,0]]), np.sum(cm))*100))
```

OUTPUT

```
TN - True Negative 83
FP - False Positive 14
FN - False Negative 15
TP - True Positive 42
Accuracy Rate: 81.16883116883116
Misclassification Rate: 18.83116883116883
```

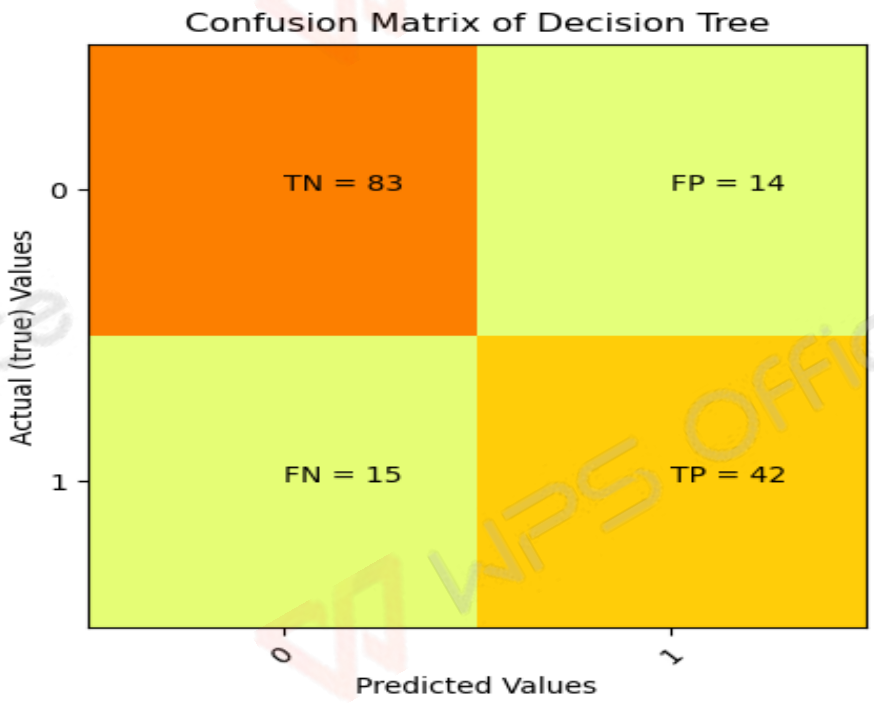
INPUT

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

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

OUTPUT



```
specificity = TN /(TN+FP)*100
print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 88.6598
```

INPUT

```
from sklearn.metrics import f1_score
print('F1_Score of Macro: ', f1_score(y_test, lr_pred)*100)
```

OUTPUT

```
F1_Score of Macro:  65.34653465346535
```

INPUT

```
print("Micro Average f1 Score is: ", f1_score(y_test, lr_pred,
average='micro')*100)
print("Macro Average f1 Score is: ", f1_score(y_test, lr_pred,
average='macro')*100)
print("Weighted Average f1 Score is: ", f1_score(y_test, lr_pred,
average='weighted')*100)
print("f1 Score on Non Weighted score is: ", f1_score(y_test, lr_pred,
average=None)*100)
```

OUTPUT

```
Micro Average f1 Score is:  77.27272727272727
Macro Average f1 Score is:  74.21916104653944
Weighted Average f1 Score is:  76.52373933045479
f1 Score on Non Weighted score is:  [83.09178744 65.34653465]
```

Classification Report of Logistic Regression:

INPUT

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n',
classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:

              precision    recall  f1-score   support

           0     0.7818    0.8866    0.8309        97
           1     0.7500    0.5789    0.6535        57

    accuracy                         0.7727       154
   macro avg     0.7659    0.7328    0.7422       154
weighted avg     0.7700    0.7727    0.7652       154
```

ROC Curve& ROC AUC

```
auc= roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of logistic Regression is ", auc)
```

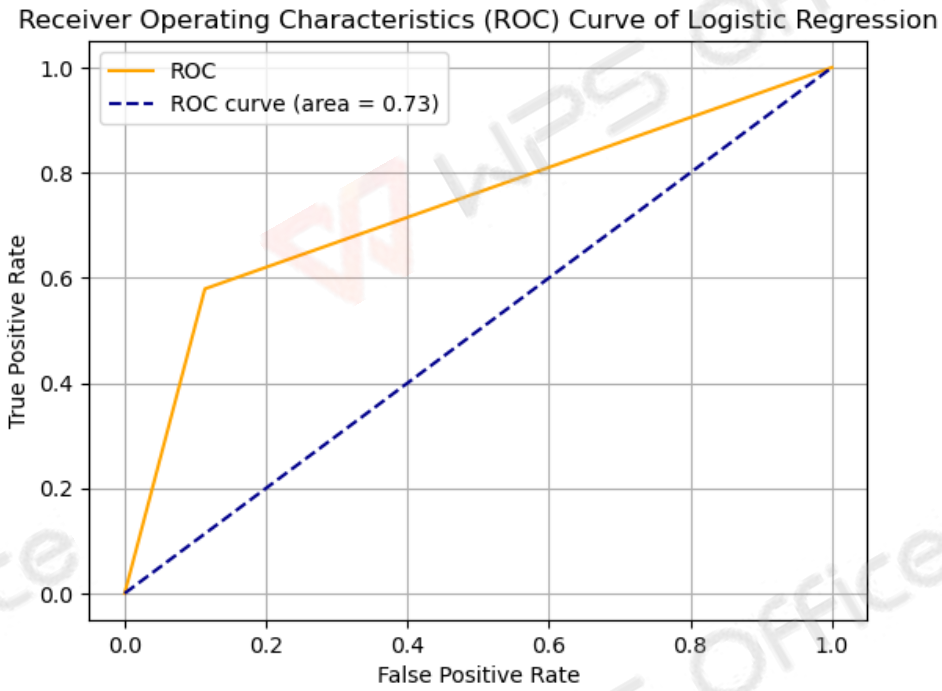
```
ROC AUC SCORE of logistic Regression is  0.7327726532826913
```

INPUT

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC
curve (area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Logistic
Regression")
plt.legend()
plt.grid()
plt.show()
```

OUTPUT



Confusion matrix of "Decision Tree"

Confusion Matrix:

```
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, dt_pred)
cm
```

OUTPUT

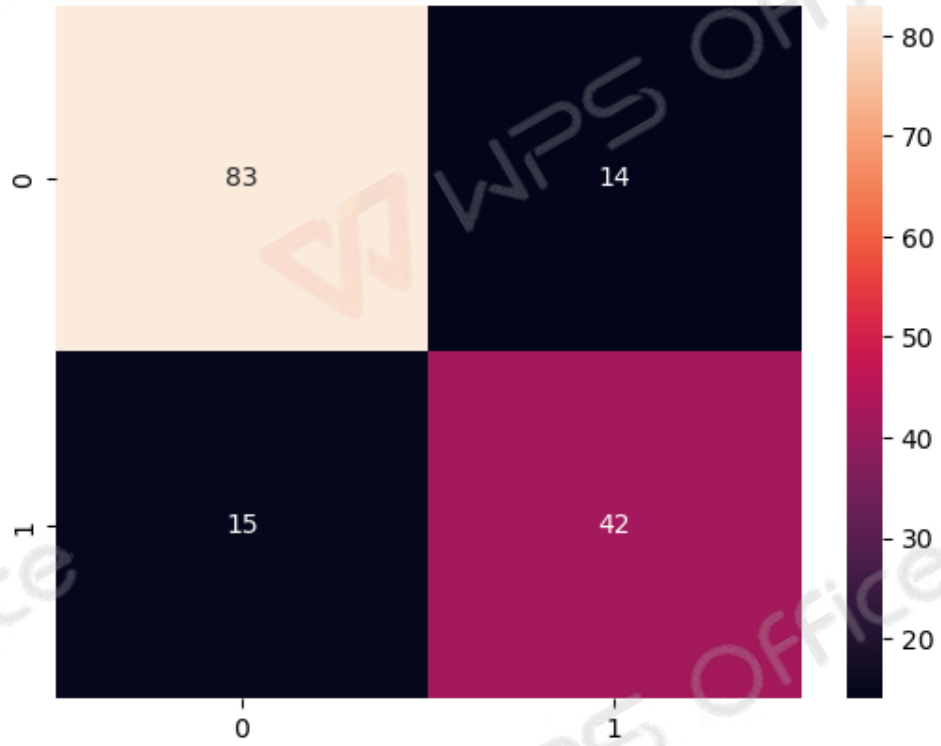
```
array([[83, 14],
       [15, 42]])
```

INPUT

```
sns.heatmap(confusion_matrix(y_test, dt_pred), annot=True, fmt="d")
```

OUTPUT

<Axes: >



INPUT

```
TN =cm[0, 0]
FP =cm[0,1]
FN = cm[1,0]
TP  = cm[1,1

TN, FP, FN, TP
```

OUTPUT

```
(83, 14, 15, 42)
```

INPUT

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, dt_pred)
```

```
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0], cm[1,1]]),
np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],
cm[1,0]]), np.sum(cm))*100))
```

OUTPUT

```
TN - True Negative 83
FP - False Positive 14
FN - False Negative 15
TP - True Positive 42
Accuracy Rate: 81.16883116883116
Misclassification Rate: 18.83116883116883
```

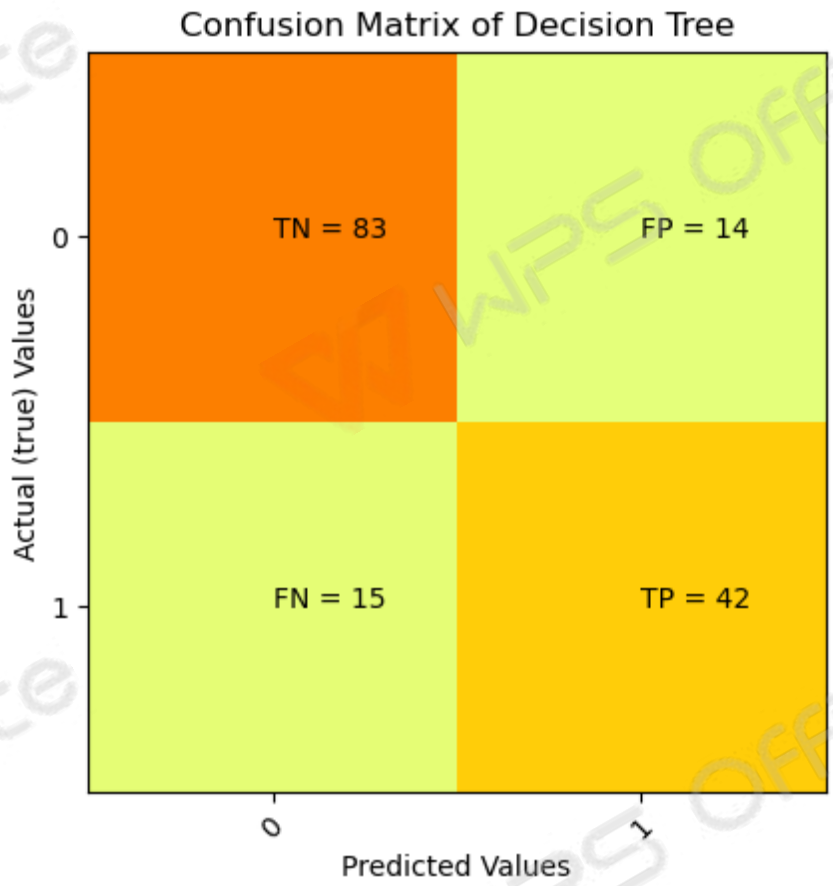
INPUT

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

plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['0', '1']
plt.title('Confusion Matrix of Decision Tree')
plt.ylabel('Actual (true) Values')
plt.xlabel('Predicted Values')
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks(tick_marks, classNames)
s = [['TN', 'FP'], ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j, i, str(s[i][j]) + " = " + str(cm[i][j]))

plt.show()
```

OUTPUT




```
auc= roc_auc_score(y_test, dt_pred)
print("ROC AUC SCORE of Decision Treeis ", auc)
```

```
ROC AUC SCORE of Decision Treeis  0.79625610417
```

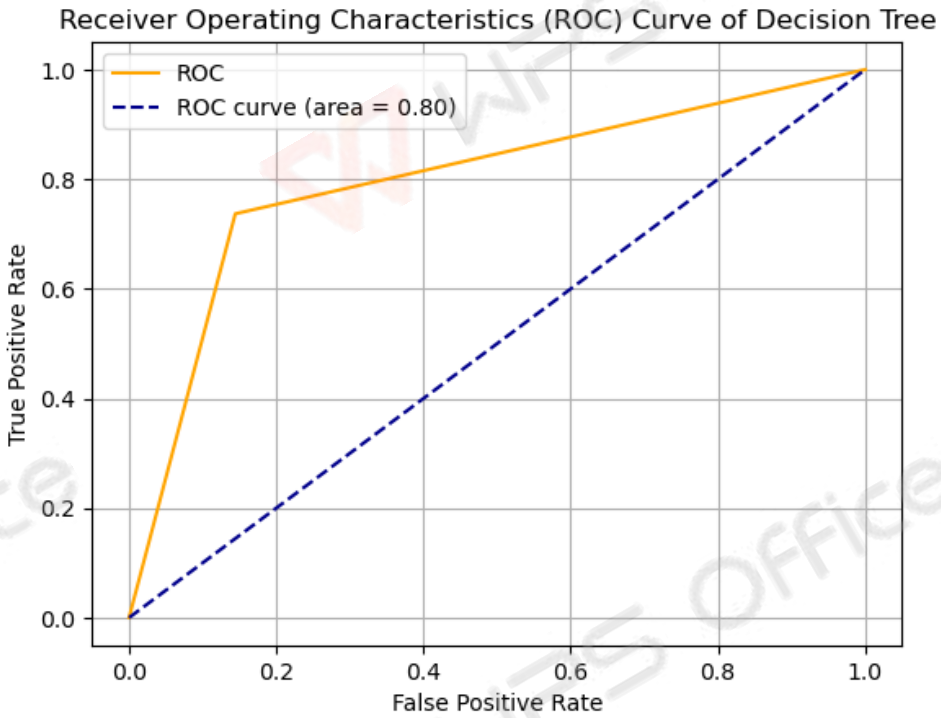
INPUT

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
plt.plot(fpr, tpr, color='orange', label="ROC")
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC
curve (area = %0.2f)' % auc(fpr, tpr))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristics (ROC) Curve of Decision
Tree")
plt.legend()
```

```
plt.grid()
plt.show()
```

OUTPUT



Receiver Operating Characteristic (ROC) curves are a useful tool in an AI-based diabetes prediction system, particularly for assessing the model's ability to discriminate between diabetes and non-diabetes cases. Here's how ROC curves are relevant:

ROC Curve: The ROC curve is a graphical representation that shows the trade-off between the model's true positive rate (recall or sensitivity) and its

false positive rate as you vary the classification threshold. The curve is created by plotting these rates for different threshold values.

AUC-ROC: The Area Under the RO C Curve (AUC-ROC) is a common metric used to quantify the overall performance of a model. An AUC-ROC value of 0.5 indicates random guessing, while a value of 1.0 indicates perfect discrimination between the two classes. In the context of a diabetes prediction system, a higher AUC-ROC indicates better discrimination between diabetic and non-diabetic cases

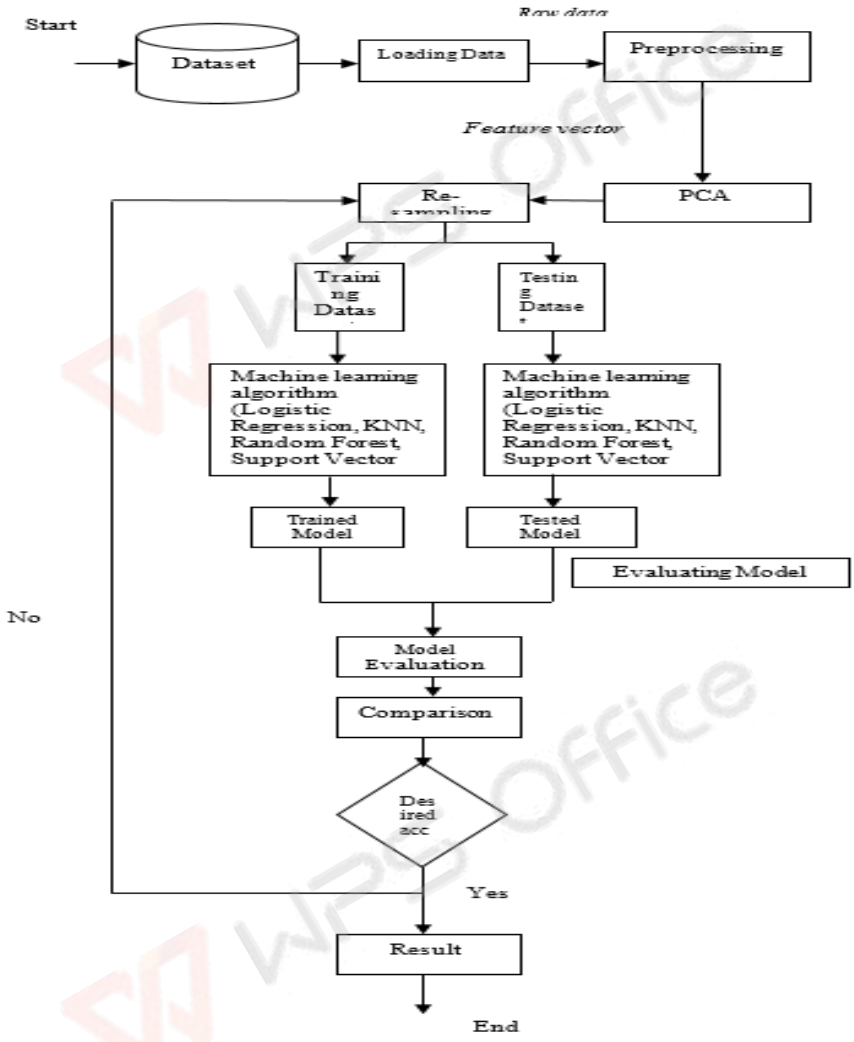
Threshold Selection: ROC curves help in selecting an appropriate classification threshold based on your system's requirements. A point on the curve that balances sensitivity and specificity can be chosen to optimize model performance.

Model Comparison: You can compare different models or variations of your model by examining their ROC curves and AUC-ROC values. A model with a higher AUC-ROC generally performs better in discriminating between diabetes and non-diabetes cases.

Imbalanced Datasets: ROC curves are particularly useful when dealing with imbalanced datasets, where one class significantly outnumbers the other. It provides insights into how well the model is handling such imbalances.

- ROC curves and AUC-ROC are valuable tools for understanding and evaluating the performance of your diabetes prediction model, especially in terms of its ability to correctly classify patients with diabetes and those without. They provide a visual representation of the model's discriminatory power, which is crucial in healthcare.

Here is the summary of the total process of AI_Based diabetes prediction model using machine learning.



CONCLUSION:

In conclusion, building an AI-based diabetes prediction system involves several crucial steps, including preprocessing of dataset, feature engineering, model selection, and evaluation. Here's a summary of key takeaways for each aspect:

Feature Engineering: Collect and preprocess a diverse dataset with relevant diabetes-related features.

- Handle missing data and outliers appropriately.
- Create new features, if necessary, based on domain knowledge
- Perform feature scaling to ensure features are on a similar scale.
- Conduct feature selection to identify the most relevant predictors.

Model Selection: Choose a variety of machine learning models, including logistic regression, decision trees, random forests, and others.

- Select models suitable for binary classification tasks in a healthcare context.
- Experiment with different algorithms to find the one that performs best on your dataset.
- Consider the interpretability ofthe model, which is crucial in healthcare applications.

Evaluation: Split the dataset into training and testing sets to assess model performance.

Utilize performance metrics like accuracy, precision, recall, F1-score, and AUC-ROC.

Be mindful of imbalanced datasets and select appropriate evaluation metrics.

Consider the trade-off between precision and recall to balance false positives and false negatives.

Use confusion matrices to gain detailed insights into model performance.

Explore ROC curves and AUC-ROC to assess the model's discriminatory power.

Regularly update and retrain the model as new data becomes available to maintain its predictive accuracy.

In practice, it's essential to iteratively refine your AI-based diabetes prediction system by experimenting with different approaches, models, and evaluation methods. The choice of features, models, and evaluation metrics should align with the specific characteristics of your dataset and the clinical goals of your prediction system. Continuous monitoring and updates are critical to ensure the system remains accurate and reliable over time.