Comparison between Logistic Regression and Decision Tree

Dataset for modeling

	Age	BMI	Glucose	BloodPressure	FamilyHistory	Diabetic
0	51.0	21.119129	192.0	95.0	No	0.0
1	43.0	25.436085	73.0	112.0	No	1.0
2	25.0	19.137683	107.0	88.0	Yes	0.0
3	66.0	NaN	80.0	99.0	Yes	0.0
4	55.0	NaN	116.0	116.0	No	0.0

This dataset likely for predicting diabetes. It contains five features and over two hundred samples which contribute for predicting a person is diabetic or not.

Age: The age of the individual, in years.

BMI: Body Mass Index. Note that there are missing values (represented as "NaN") in rows 3 and 4.

Glucose: A measure of blood glucose level.

BloodPressure: A measure of blood pressure.

FamilyHistory: Indicates whether the individual has a family history of diabetes ("Yes" or "No").

Diabetic: The target variable, indicating if the person is diabetic. A value of 0.0 likely means "No" and 1.0 means "Yes".

Counting null values

	0
Age	19
BMI	19
Glucose	17
BloodPressure	20
FamilyHistory	0
Diabetic	20

Detailed overview of Dataset before preprocessing

	Age	BMI	Glucose	BloodPressure	FamilyHistory	Diabetic
count	181.000000	181.000000	183.000000	180.000000	200	180.000000
unique	NaN	NaN	NaN	NaN	2	NaN
top	NaN	NaN	NaN	NaN	No	NaN
freq	NaN	NaN	NaN	NaN	109	NaN
mean	53.520862	34.119600	147.024905	94.879971	NaN	0.609337
std	25.851857	11.962385	54.333459	24.233665	NaN	0.752135
min	20.000000	18.077110	70.000000	60.000000	NaN	0.000000
25%	34.000000	25.436085	112.000000	79.000000	NaN	0.000000
50%	51.000000	33.990356	139.000000	92.000000	NaN	0.500000
75%	65.000000	39.368697	173.500000	106.250000	NaN	1.000000
max	135.127608	71.684465	318.955757	171.039473	NaN	2.968063

Converting all categorical to numeric

converting categorical data to numeric is important for model designing because code works only on numerical data

here except FamilyHistory column I'm converting remaining columns to numerical because I have to convert categorical like 'YES' and 'No' should be encoded to numerical values later.

```
for col in df.columns:
  if(col!='FamilyHistory'):
    df[col]=pd.to_numeric(df[col],errors='coerce')
df
```

	Age	BMI	Glucose	BloodPressure	FamilyHistory	Diabetic
0	51.0	21.119129	192.0	95.000000	No	0.0
1	43.0	25.436085	73.0	112.000000	No	1.0
2	25.0	19.137683	107.0	88.000000	Yes	0.0
3	66.0	NaN	80.0	99.000000	Yes	0.0
4	55.0	NaN	116.0	116.000000	No	0.0
...	...	...	...	...	...	...
195	46.0	NaN	99.0	105.000000	Yes	0.0
196	74.0	22.619721	195.0	97.000000	No	0.0
197	41.0	38.112320	NaN	90.000000	Yes	0.0
198	66.0	NaN	154.0	171.039473	No	0.0
199	58.0	34.902176	98.0	114.000000	No	0.0

Converting categorical values of Familycolumn to numerical values

```
df['FamilyHistory'] = df['FamilyHistory'].map({'Yes': 1, 'No': 0})
df
```

	Age	BMI	Glucose	BloodPressure	FamilyHistory	Diabetic
0	51.0	21.119129	192.0	95.000000	0	0.0
1	43.0	25.436085	73.0	112.000000	0	1.0
2	25.0	19.137683	107.0	88.000000	1	0.0
3	66.0	NaN	80.0	99.000000	1	0.0
4	55.0	NaN	116.0	116.000000	0	0.0
...	...	...	...	...	...	...
195	46.0	NaN	99.0	105.000000	1	0.0
196	74.0	22.619721	195.0	97.000000	0	0.0
197	41.0	38.112320	NaN	90.000000	1	0.0
198	66.0	NaN	154.0	171.039473	0	0.0
199	58.0	34.902176	98.0	114.000000	0	0.0

200 rows × 6 columns

Here I'm mapping categorical values like 'Yes' or 'No' to numerical values '1' and '0'

Yes – '1'

No – '0'

Let's Do some EDA

Data distribution

```
import seaborn as sns
import matplotlib.pyplot as plt
for col in df.columns:
  sns.histplot(df[col],kde=True)
  plt.title(f'Distribution of {col}')
  plt.xlabel(col)
  plt.ylabel('frequency')
  plt.show()
```

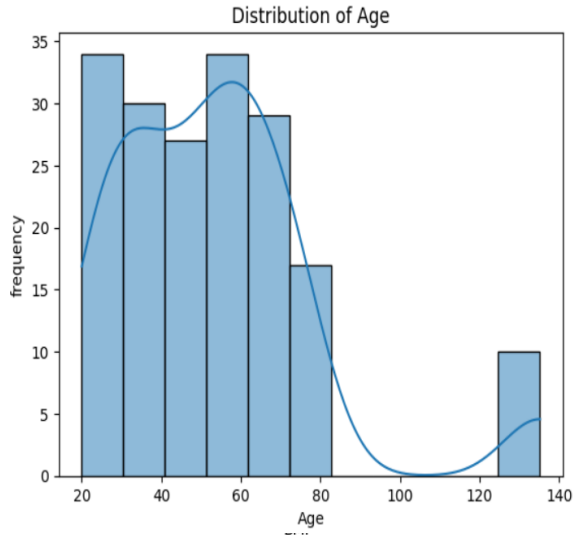
`df.select_dtypes(include=['number'])`: Skips non-numeric columns to prevent plotting errors.

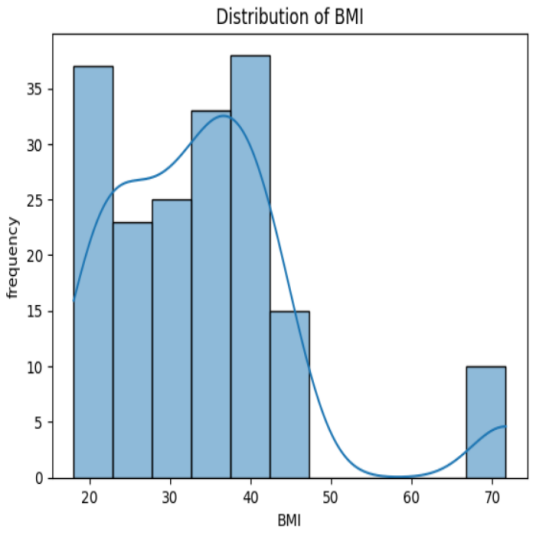
`plt.figure(figsize=(8, 4))`: Sets a consistent size for all plots.

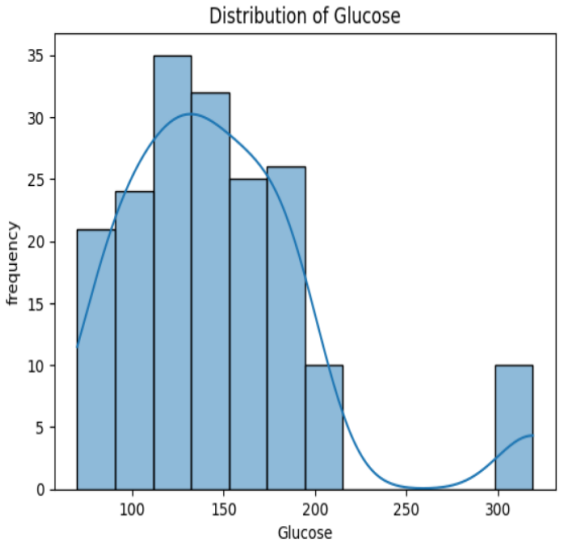
`bins=30`: Controls histogram granularity (adjustable).

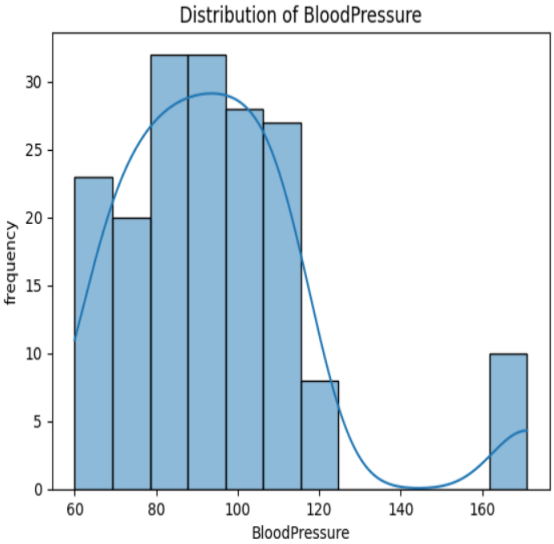
`plt.tight_layout()`: Prevents label overlap.

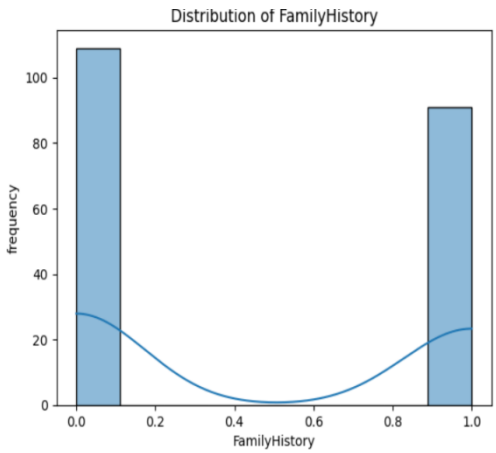
Optional: Add `log_scale=True` for skewed data.

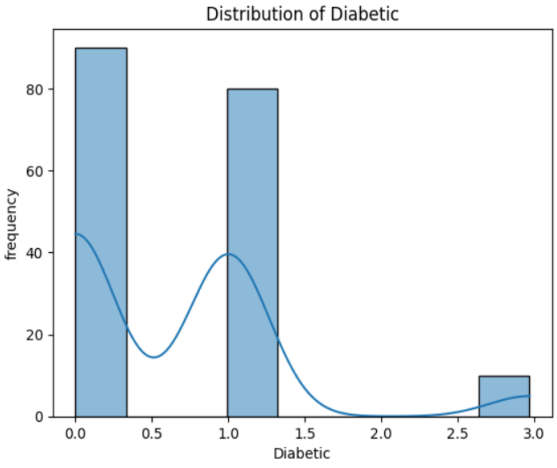












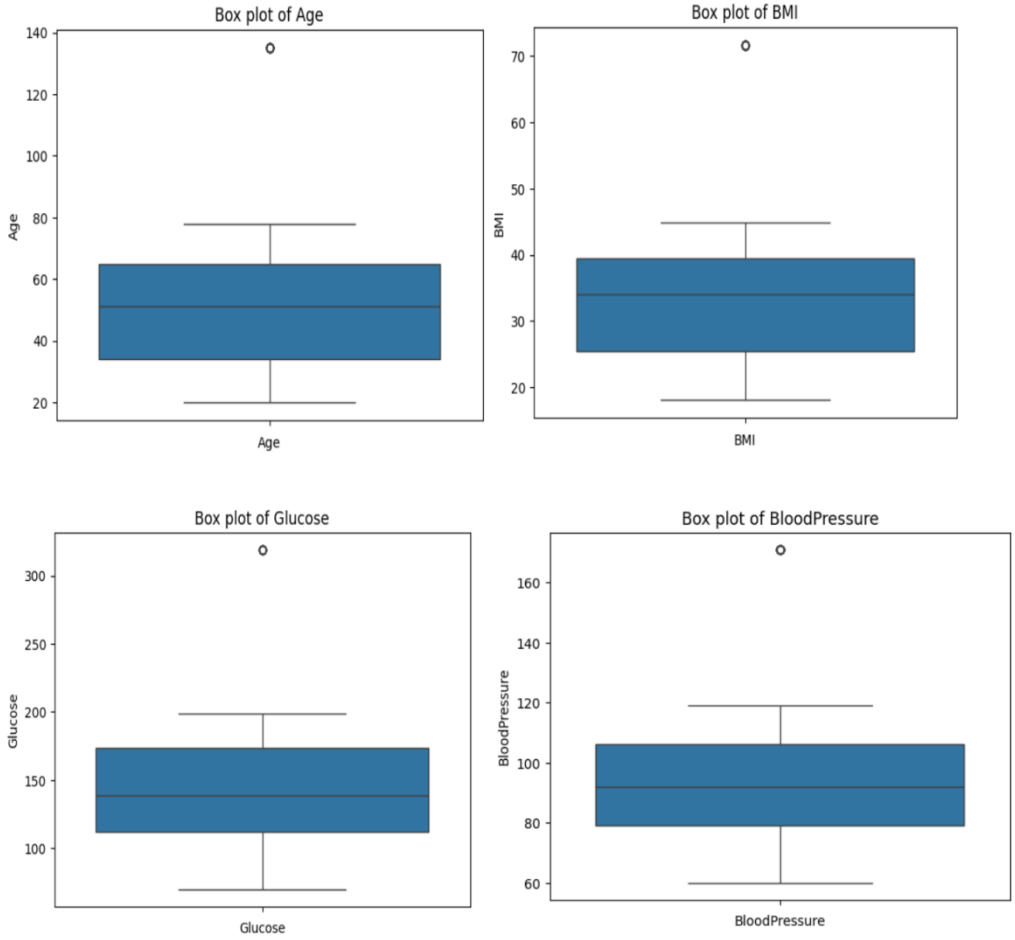
Here we can see distribution of family history its either 0 or 1 so here we don't need to do any changes like diabetic columns cause it contains some outliers which are to be removed we can get clearification with box plots which are used for identifying outliers.

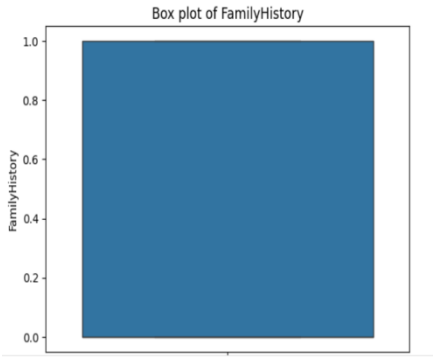
Box plot for identifying outliers

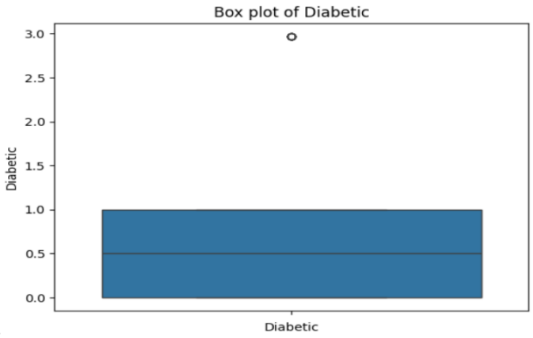
Source code for box plot

```
for col in df.columns:
  sns.boxplot(df[col])
  plt.title(f'Box plot of {col}')
  plt.xlabel(col)

  plt.show()
```

here we are traversing through each column to get box plot for each feature inorder to identify outliers.







Here we can see that each and every features has outliers except familyhistory that's what I said in above statement. These outliers may negatively impact on prediction so inorder to get a better prediction we must remove these or replace these.

But in Diabetic column there are some outliers which shouldn't be replaced with mean or median or mode because it is an output column doing replacing may effects output.

Replacing NAN with values

```
df['FamilyHistory'].fillna(df['FamilyHistory'].mode()[0],inplace=True)
```

here i'm replacing NAN to mode in FamilyHistory column I am doing these because it has very less feature importance in prediction.

```
for col in df.columns:
  if(col!='FamilyHistory' and col!='Diabetic'):
    df[col].fillna(df[col].mean(),inplace=True)
```

except FamilyHistory and Diabetic columns I am replacing each and every NAN to mean values corresponding to their features mean

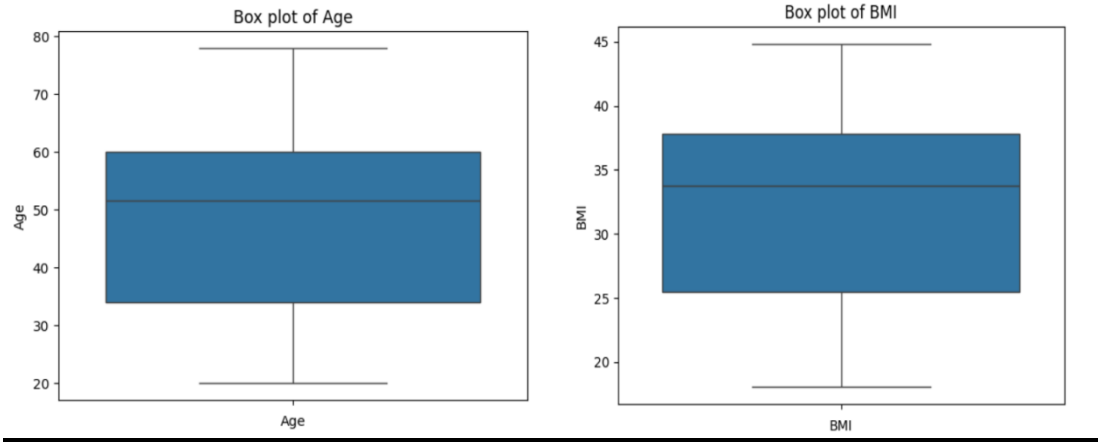
```
df = df.dropna(subset=['Diabetic'])
```

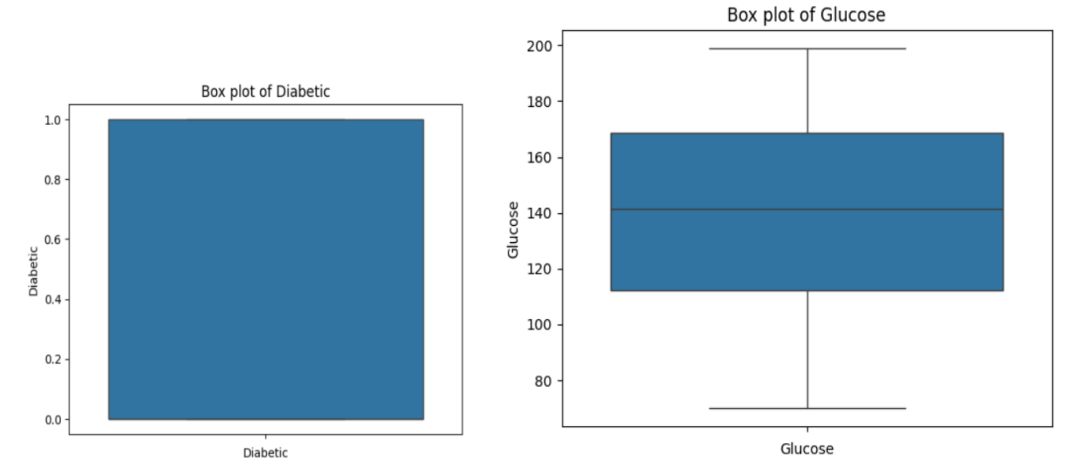
dropping rows which contains NAN

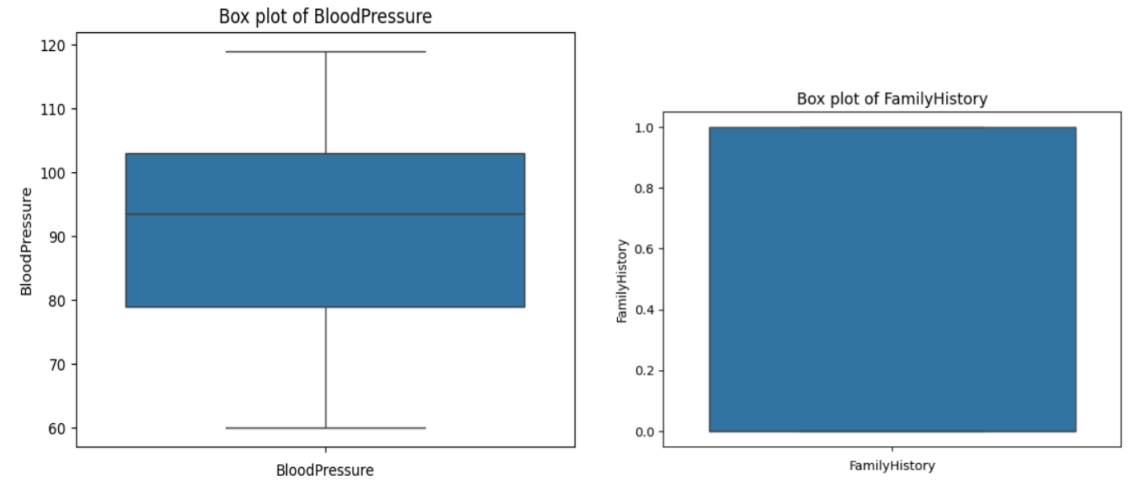
```
for col in df.columns:
  if(col!='FamilyHistory'):
    Q1=df[col].quantile(0.25)
    Q3=df[col].quantile(0.75)
    IQR=Q3-Q1
    lower_bound=Q1-1.5*IQR
    upper_bound=Q3+1.5*IQR
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
for col in df.columns:
  sns.boxplot(df[col])
  plt.title(f'Box plot of {col}')
  plt.xlabel(col)
  plt.show()
```

in this code I’m removing outliers of every features and visualizing box plots to ensure no outliers are present.

Box plot after removing outliers

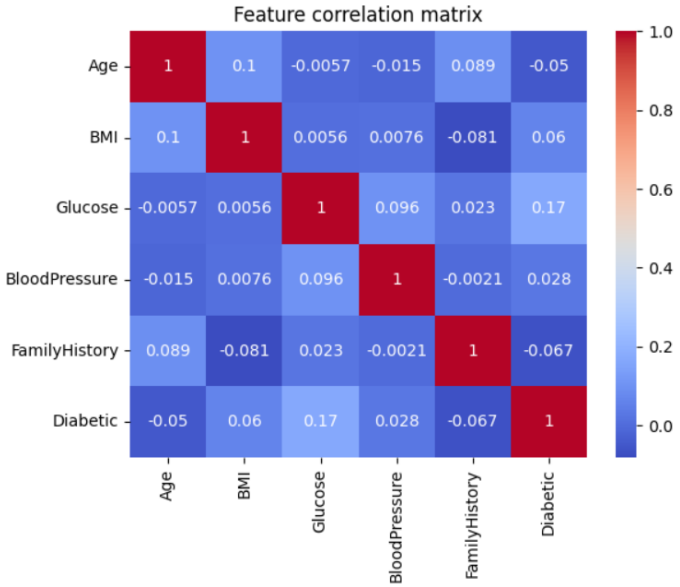






Correlation Matrix

```
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')
plt.title('Feature correlation matrix')
plt.show()
```



Statistics after data preprocessing

	Age	BMI	Glucose	BloodPressure	FamilyHistory	Diabetic
count	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000
mean	52.528080	33.725072	147.774710	95.214513	0.450000	0.609337
std	23.847127	11.064624	51.994973	23.729790	0.498881	0.752135
min	20.000000	18.077110	70.000000	60.000000	0.000000	0.000000
25%	34.750000	25.708063	114.000000	79.750000	0.000000	0.000000
50%	52.528080	33.725072	146.887355	95.000000	0.000000	0.500000
75%	62.000000	38.168955	172.000000	105.250000	1.000000	1.000000
max	135.127608	71.684465	318.955757	171.039473	1.000000	2.968063

Modeling

Logistic Regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
X = df.drop('Diabetic', axis=1)
y = df['Diabetic']
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2,random_state=42)
model = LogisticRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print(accuracy)
print('Confusion Matrix')
cm=confusion_matrix(y_test,y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not
Diabetic', 'Diabetic'], yticklabels=['Not Diabetic', 'Diabetic'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
print('Classification Report')
print(classification_report(y_test,y_pred))
```

Explaination :

# 1. Import train-test split utility
from sklearn.model_selection import train_test_split

# 2. Import logistic regression model
from sklearn.linear_model import LogisticRegression

# 3. Import evaluation metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# 4. Set feature variables by dropping the target column
X = df.drop('Diabetic', axis=1)

# 5. Set the target variable
y = df['Diabetic']

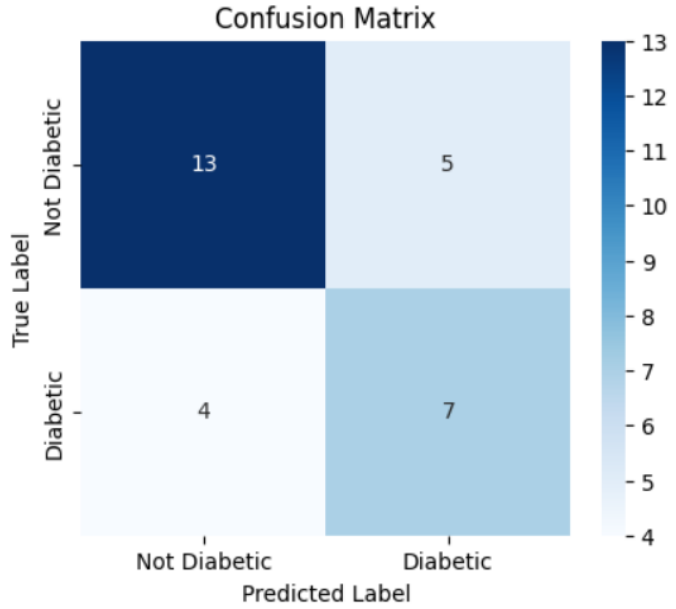
6. Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.2, random_state=42)

```
# 7. Create a logistic regression model
model = LogisticRegression()

# 8. Train the model using training data
model.fit(X_train, y_train)

# 9. Predict the labels for the test data
y_pred = model.predict(X_test)

# 10. Calculate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)

# 11. Print the accuracy score
print(accuracy)

# 12. Print heading for confusion matrix
print('Confusion Matrix')

# 13. Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# 14. Set the size of the confusion matrix plot
plt.figure(figsize=(5, 4))

# 15. Plot the heatmap of confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=['Not Diabetic', 'Diabetic'],
        yticklabels=['Not Diabetic', 'Diabetic'])

# 16. Label the x-axis
plt.xlabel('Predicted Label')

# 17. Label the y-axis
plt.ylabel('True Label')

# 18. Set the title of the heatmap
plt.title('Confusion Matrix')

# 19. Display the heatmap
plt.show()

# 20. Print heading for classification report
print('Classification Report')

# 21. Print precision, recall, and F1-score
print(classification_report(y_test, y_pred))
```

output:

```
accuracy:0.6896551724137931
```

confusion matrix



By this we can say logistic regression model for this purpose predicted 13 correctly as non diabetic and misclassifies not diabetic as diabetic 4 times out of 17 predicitons.same for diabetic as well it has predicted 7 correctly and 5 missfications out of 12 predicitions.

Classification Report

	precision	recall	f1-score	support
0.0	0.76	0.72	0.74	18
1.0	0.58	0.64	0.61	11
accuracy			0.69	29
macro avg	0.67	0.68	0.68	29
weighted avg	0.70	0.69	0.69	29

feature importance

```
import pandas as pd
coeff_df =
pd.DataFrame({'Feature':X.columns,'Coefficient':model.coef_[0]})
print(coeff_df)
print('Intercept:',model.intercept_[0])
```

output:

```
          Feature  Coefficient
0             Age    -0.000253
1             BMI     0.003822
2         Glucose     0.001162
3   BloodPressure    -0.000596
4   FamilyHistory    -0.299330
Intercept: -0.2026510516781127
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
treemodel = DecisionTreeClassifier(random_state=42)
treemodel.fit(X_train,y_train)
y_pred_tree = treemodel.predict(X_test)
print(classification_report(y_test,y_pred_tree))
accuracy = accuracy_score(y_test,y_pred_tree)
print(accuracy)
cm=confusion_matrix(y_test,y_pred_tree)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not
Diabetic', 'Diabetic'], yticklabels=['Not Diabetic', 'Diabetic'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

explanation:

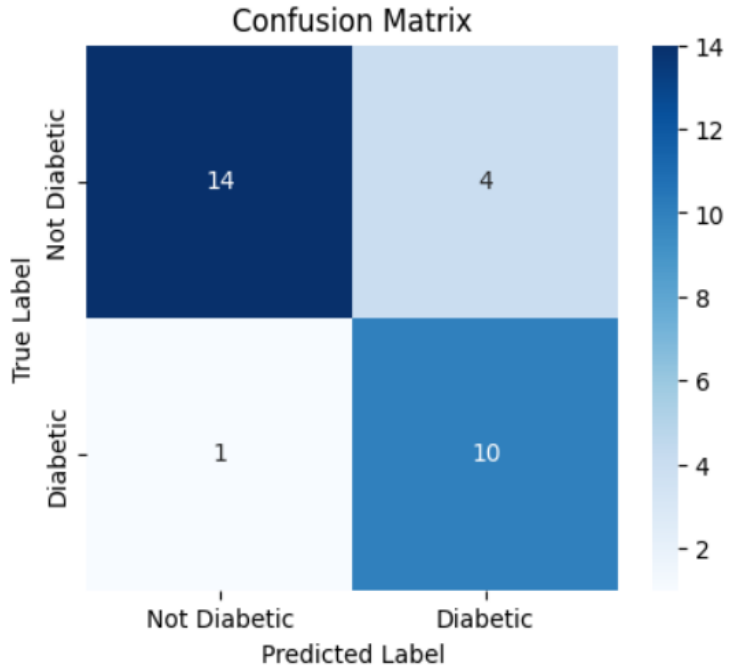
```
# 1. Import the Decision Tree classifier
from sklearn.tree import DecisionTreeClassifier

# 2. Initialize the model with a fixed random state for reproducibility
treemodel = DecisionTreeClassifier(random_state=42)

# 3. Train the model using training data
treemodel.fit(X_train, y_train)

# 4. Predict target values on test data
y_pred_tree = treemodel.predict(X_test)

# 5. Print a classification report (precision, recall, F1-score, support)
print(classification_report(y_test, y_pred_tree))

# 6. Calculate and print the accuracy score
accuracy = accuracy_score(y_test, y_pred_tree)
print(accuracy)

# 7. Generate confusion matrix
cm = confusion_matrix(y_test, y_pred_tree)

# 8. Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=['Not Diabetic', 'Diabetic'],
        yticklabels=['Not Diabetic', 'Diabetic'])

# 9. Add axis labels and title to the heatmap
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
```

plt.show()

OUTPUT:

```
              precision    recall  f1-score   support

         0.0       0.93      0.78      0.85        18
         1.0       0.71      0.91      0.80        11

    accuracy                           0.83        29
   macro avg       0.82      0.84      0.82        29
weighted avg       0.85      0.83      0.83        29
```

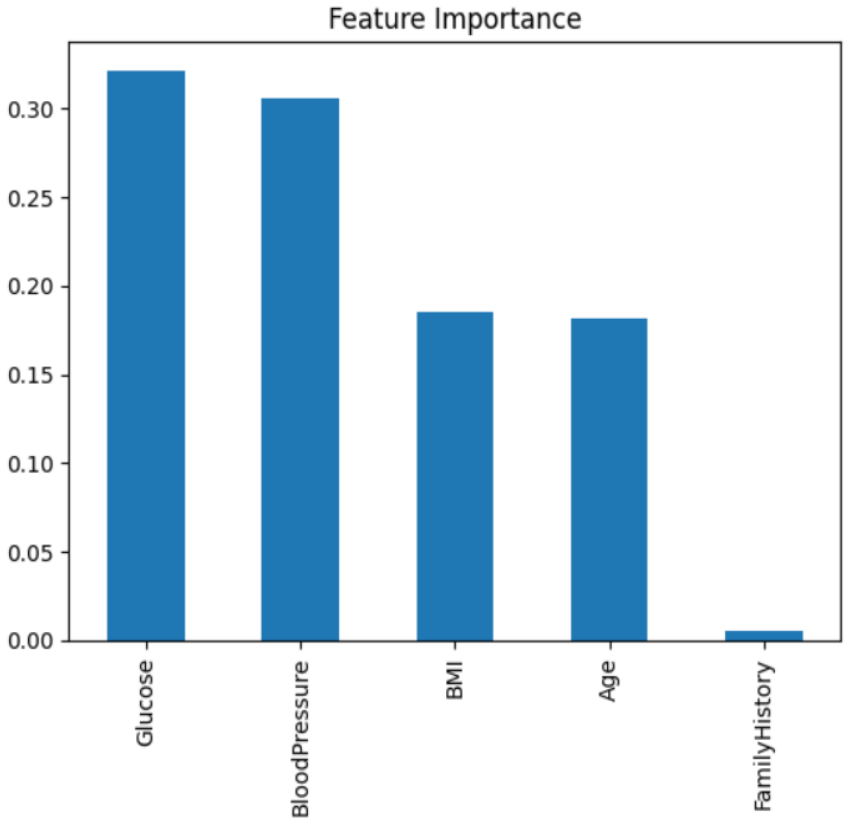
Acurracy : 0.8275862068965517



Feature importance:

Which tells about how much each feature contributing in prediciton

```
feature_importance = treemodel.feature_importances_
feature_name = X.columns
pd.Series(feature_importance,index=feature_name).sort_values(ascending=
False).plot(kind='bar')
plt.title('Feature Importance')
plt.show()
```



SOME MAIN DIFFERENCES BETWEEN LOGISTIC RERESSION AND DECISION TREE

Metric / Aspect	Logistic Regression	Decision Tree Classifier
Accuracy	0.69 (≈ 69%)	0.83 (≈ 83%)
Precision (Class 0)	0.76	0.93
Recall (Class 0)	0.72	0.78
F1-Score (Class 0)	0.74	0.85
Precision (Class 1)	0.58	0.71
Recall (Class 1)	0.64	0.91
F1-Score (Class 1)	0.61	0.80
Macro F1-Score	0.68	0.82
Weighted F1-Score	0.69	0.83
Confusion Matrix (0 label)	13 correct, 5 misclassified	14 correct, 4 misclassified
Confusion Matrix (1 label)	7 correct, 4 misclassified	10 correct, 1 misclassified
Feature Importance Basis	Model Coefficients (linear contribution)	Node Splitting (information gain/impurity reduction)
Feature Scaling Required	Yes (for better interpretation)	No

Metric / Aspect	Logistic Regression	Decision Tree Classifier
Handles Nonlinearity	✗No	✓Yes
Interpretability	High (for linear relations)	High (but can become complex in deep trees)

1. Performance

Decision Tree clearly outperforms **Logistic Regression** in:

Accuracy

F1-score (especially for diabetic class – class 1)

Better handling of class imbalance and non-linear patterns

2. Feature Contribution

Logistic Regression:

Uses coefficients; interpretation is linear and depends on scale

Most negatively impacting feature: `FamilyHistory`

Decision Tree:

Uses impurity-based importance

Shows clear ranking of which features are truly influential in splits

3. Interpretability

Logistic Regression is more interpretable if the relationship is **linear**.

Decision Tree is better if you suspect **interactions or thresholds** between features (e.g., BMI > threshold).

4. Use Case Fit

Logistic Regression is best for:

Baseline modeling

Well-behaved data (linearly separable, low noise)

Decision Tree is better for:

Nonlinear decision boundaries

Capturing interactions without feature engineering

Conclusion:

Based on the evaluation metrics and confusion matrix:

Decision Tree Classifier is the better model for this dataset, especially for correctly identifying diabetic patients (class 1), which is often more **critical** in real-life healthcare applications.

Logistic Regression can still be useful as a **benchmark model** or if interpretability in terms of feature weight is needed.