AIES_logistic_regression

October 20, 2024

Diabetes Prediction using Logistic Regression

Dataset : **Kaggle-UCIrvine**

By : Hrishikesh Iyer

1 Basic Data Ops

[3]:
```
import pandas as pd
```

1.1 Fetching the dataset into a dataframe

[5]:
```
df = pd.read_csv("cleaned_data_logistic_regression.csv")
```

1.2 Displaying the dataframe

Observations :
– All are numeric values , so no need to reinterpret
– *Outcome* will be Y (the dependant var) , others will be in X vector

[7]:
```
df.info()
```

```
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

1.3 Checking for NULL values

Observations :
– no NULL values

[9]:
```
df.isnull().sum()
```

[9]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

2 Logistic Regression

2.1 Dependant and Independant Variables

– *Outcome* column will be the dependant variable => Y
– Others would be part of the independant variables vector $ X $ where $X = <X1, X2...Xn>$

[12]:
```
X = df.drop(columns = "Outcome")
Y = df["Outcome"]
```

2.2 Splitting the Dataframe into Training and Testing Sets

– Training set = 70%
– Test set = 30%

[14]:
```
from sklearn.model_selection import train_test_split
```

[15]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=21)
```

2.3 Normalisation/Scaling

Using Standard Scaler :

$z = \frac{x-\mu}{\sigma}$

[17]:
```
from sklearn.preprocessing import StandardScaler
```

[18]:
```
scaler = StandardScaler()
```

[19]:
```
X_train_scaled = scaler.fit_transform(X_train)
```

[20]:
```
X_test_scaled = scaler.transform(X_test)
```

2.4 Logistic Regression without Regularisation

2.4.1 Preparing the Model

[23]:
```
from sklearn.linear_model import LogisticRegression
```

[24]:
```
log_reg = LogisticRegression(random_state = 0).fit(X_train_scaled, Y_train)
```

[25]:
```
log_reg.predict(X_train_scaled)
```

[25]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1])
```

2.4.2 Accuracy on Training Set

[27]:
```
log_reg.score(X_train_scaled, Y_train)
```

[27]: 0.7877094972067039

2.4.3 Accuracy on Test Set

[29]:
```
log_reg.score(X_test_scaled, Y_test)
```

[29]: 0.7402597402597403

2.5 Logistic Regression with Regularisation

Avoiding overfitting by initialising hyper-parameter C

[31]:
```
log_reg1 = LogisticRegression(random_state = 0, C = 0.55, fit_intercept = True).
    ↪fit(X_train_scaled, Y_train)
```

[32]:
```
log_reg1.score(X_train_scaled, Y_train)
```

[32]: 0.7895716945996276

[33]:
```
log_reg1.score(X_test_scaled, Y_test)
```

[33]: 0.7402597402597403

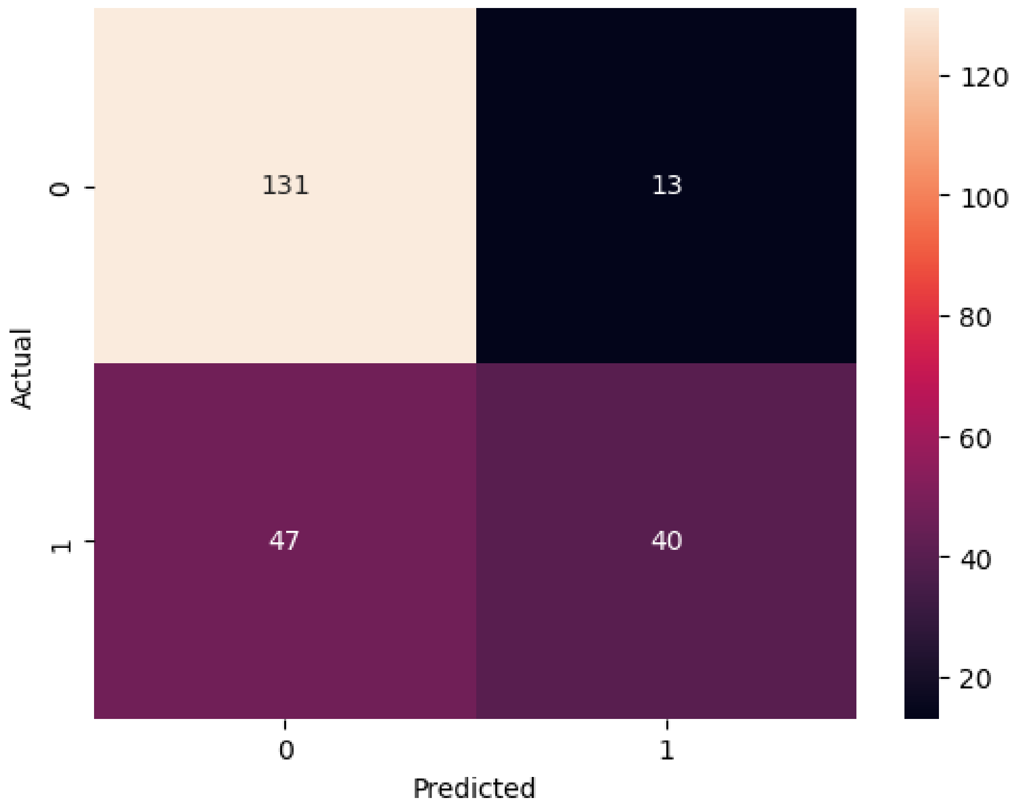
[34]:
```
print(log_reg1.coef_)
```

```
[[ 0.38017565  1.16278816 -0.22176045  0.062958   -0.00353279  0.69927688
   0.24102053  0.23311289]]
```

3 Visualising the Prediction

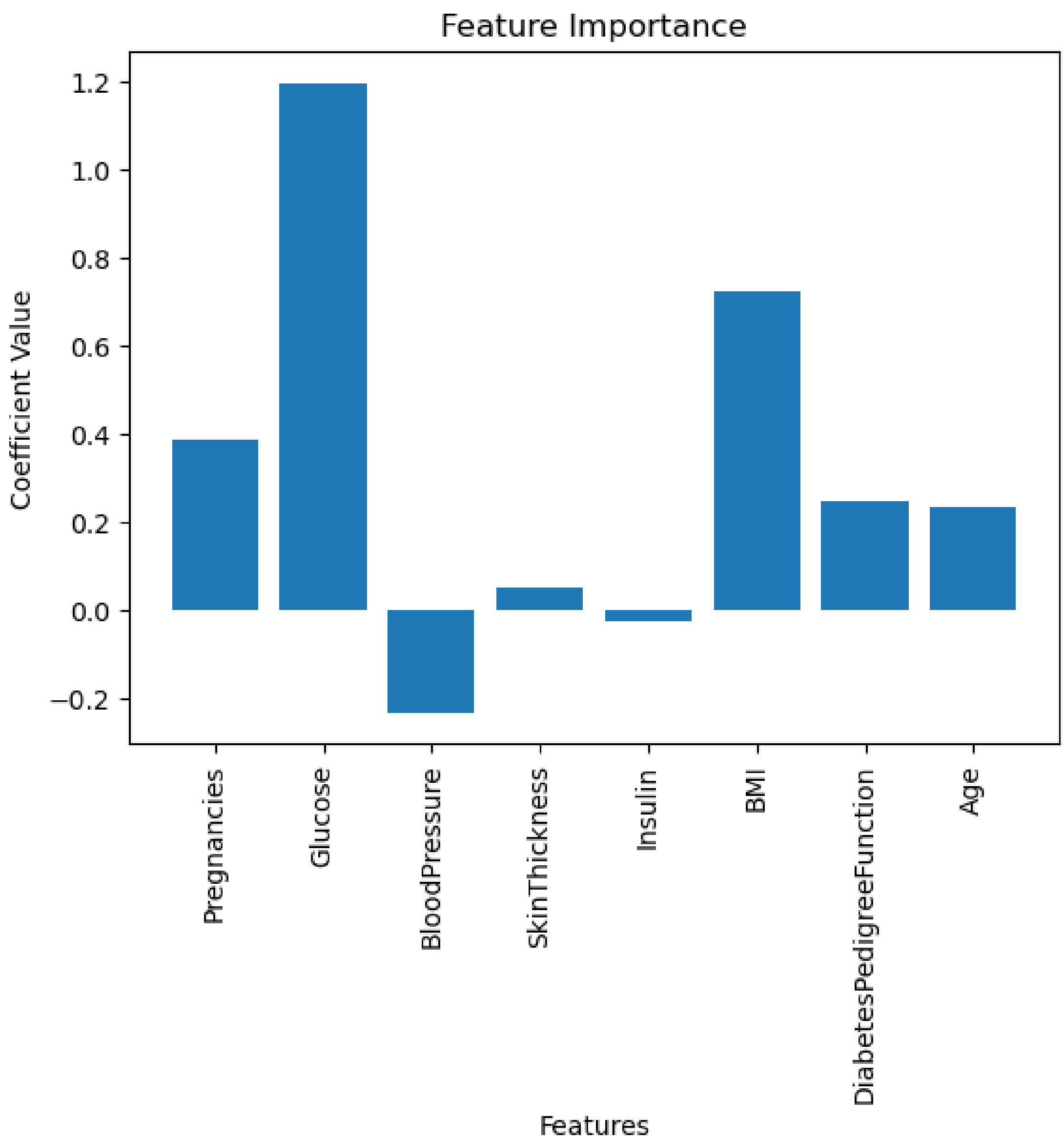
3.1 Confusion Matrix

[37]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(Y_test, log_reg.predict(X_test_scaled))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



3.2 Feature Importance

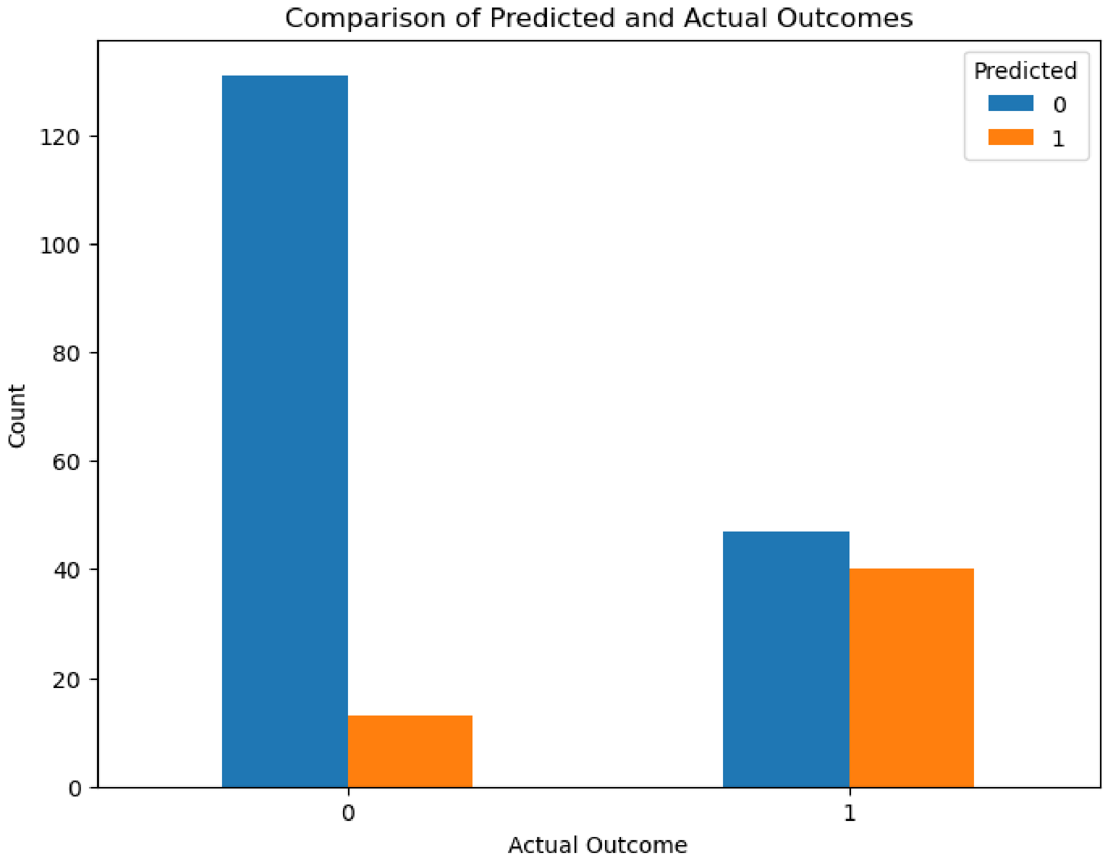
[39]:
```
coefficients = log_reg.coef_[0]
feature_names = X.columns
plt.bar(feature_names, coefficients)
plt.xticks(rotation=90)
plt.xlabel('Features')

plt.ylabel('Coefficient Value')
plt.title('Feature Importance')
plt.show()
```



3.3 Grouped Bar Chart

[41]:
```
predictions = log_reg.predict(X_test_scaled)
df_comparison = pd.DataFrame({'Actual': Y_test, 'Predicted': predictions})
counts = df_comparison.groupby(['Actual', 'Predicted']).size().unstack()

counts.plot(kind='bar', figsize=(8, 6))
plt.title('Comparison of Predicted and Actual Outcomes')
plt.xlabel('Actual Outcome')
plt.ylabel('Count')
plt.xticks(rotation=0)
```

```
plt.legend(title='Predicted')
plt.show()
```



4 Conclusion + Model Analysis

4.1 Accuracy

– The overall correctness of the model
Calculated as

$\frac{TP+TN}{TP+TN+FP+FN}$

[44]:
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test, log_reg.predict(X_test_scaled))
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7402597402597403
```

4.2 Precision

– Proportion of TP among TP+FP
Calculated as

$\frac{TP}{TP+FP}$

[46]:
```
from sklearn.metrics import precision_score
precision = precision_score(Y_test, log_reg.predict(X_test_scaled))
print(f"Precision: {precision}")
```

```
Precision: 0.7547169811320755
```

4.3 Recall/Sensitivity

– Proportion of TP among actual positives Calculated as

$\frac{TP}{TP+FN}$

[48]:
```
from sklearn.metrics import recall_score
recall = recall_score(Y_test, log_reg.predict(X_test_scaled))
print(f"Recall: {recall}")
```

```
Recall: 0.45977011494252873
```

4.4 F1 score

– Harmonic mean of *precision* and *recall* Calculated as

$2 * \frac{precision*recall}{precision+recall}$

[50]:
```
from sklearn.metrics import f1_score
f1 = f1_score(Y_test, log_reg.predict(X_test_scaled))
print(f"F1-Score: {f1}")
```

```
F1-Score: 0.5714285714285714
```

Diabetes

October 20, 2024

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Upload data

[2]:
```python
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving cleaned_data_logistic_regression.csv to
cleaned_data_logistic_regression.csv
```

[3]:
```python
data = pd.read_csv('cleaned_data_logistic_regression.csv')
data.head()
```

[3]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness   Insulin   BMI  \
0            6    148.0           72.0        35.0000  177.9000  33.6
1            1     85.0           66.0        29.0000   41.5995  26.6
2            8    183.0           64.0        12.9995  528.0000  23.3
3            1     89.0           66.0        23.0000   94.0000  28.1
4            0    137.0           40.0        35.0000  168.0000  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Basic Operations

[4]:
```python
data.describe()
```

[4]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  121.616535      72.044260      28.861932  164.817888
std       3.369578   30.722312      13.142954      12.748930  146.019818
```

```
min        0.000000   44.000000      24.000000      7.000000   14.000000
25%        1.000000   99.000000      64.000000     20.000000   64.000000
50%        3.000000  117.000000      72.000000     29.000000  125.400000
75%        6.000000  141.000000      80.000000     35.000000  177.900000
max       17.000000  199.000000     122.000000     99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    32.326348                  0.471876   33.240885    0.348958
std      6.966415                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.375000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[5]:
```
data.info()
```

```
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

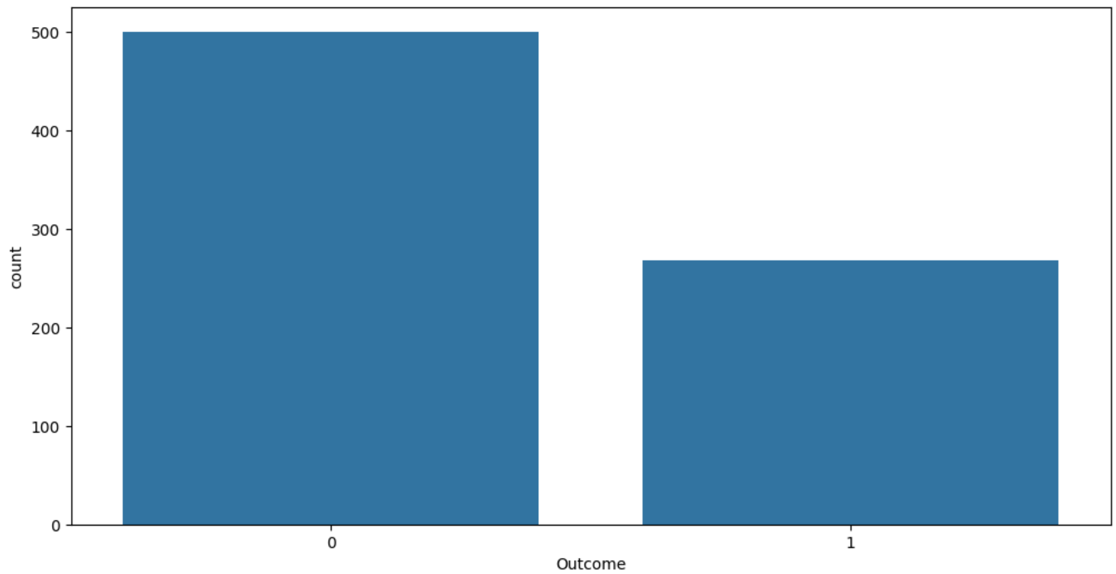
[6]:
```
data.isna().sum()
```

[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[7]:
```
data.duplicated().sum()
```

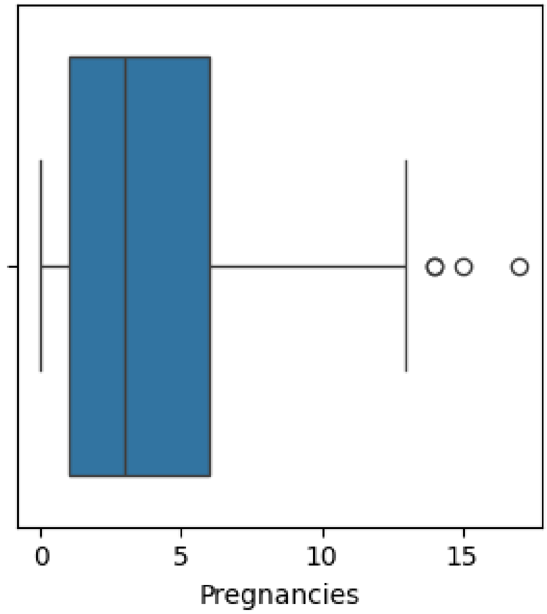
[7]: 0

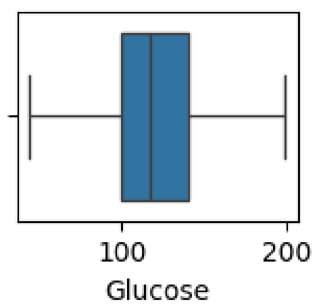
[8]:
```
plt.figure(figsize = (12,6))
sns.countplot(x = 'Outcome',data = data)
plt.show()
```

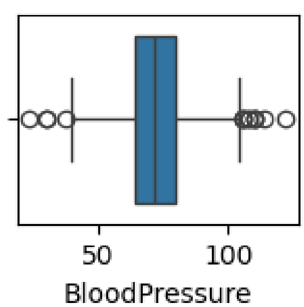


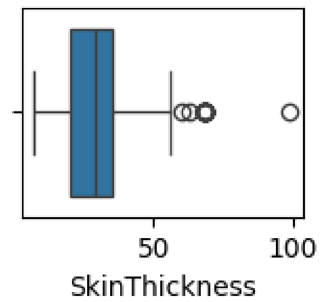
Outliers

[9]:
```
plt.figure(figsize = (12,12))
for i,col in␣
↪enumerate(['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','Diabete
↪
  plt.subplot(3,3,i+1)
  sns.boxplot(x = col,data = data)
  plt.show()
```

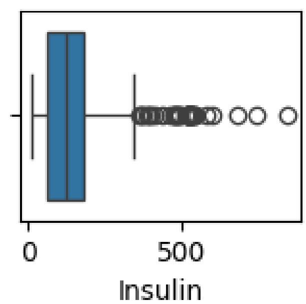

0
5
10
15
Pregnancies


100
200
Glucose

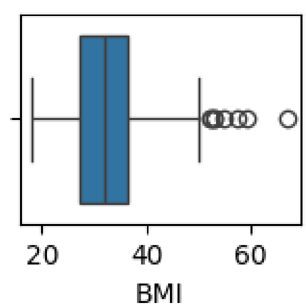

50
100
BloodPressure



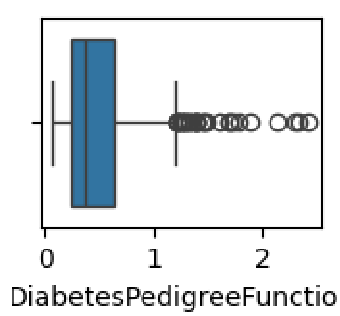
50
100
SkinThickness



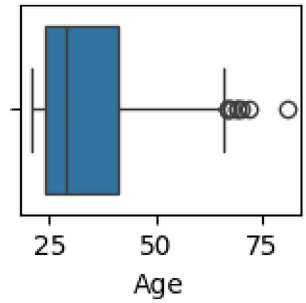
0
500
Insulin

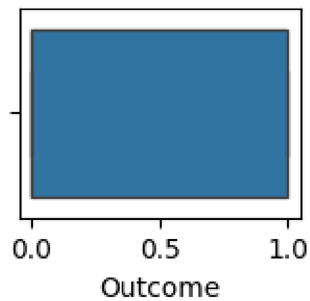


20
40
60
BMI



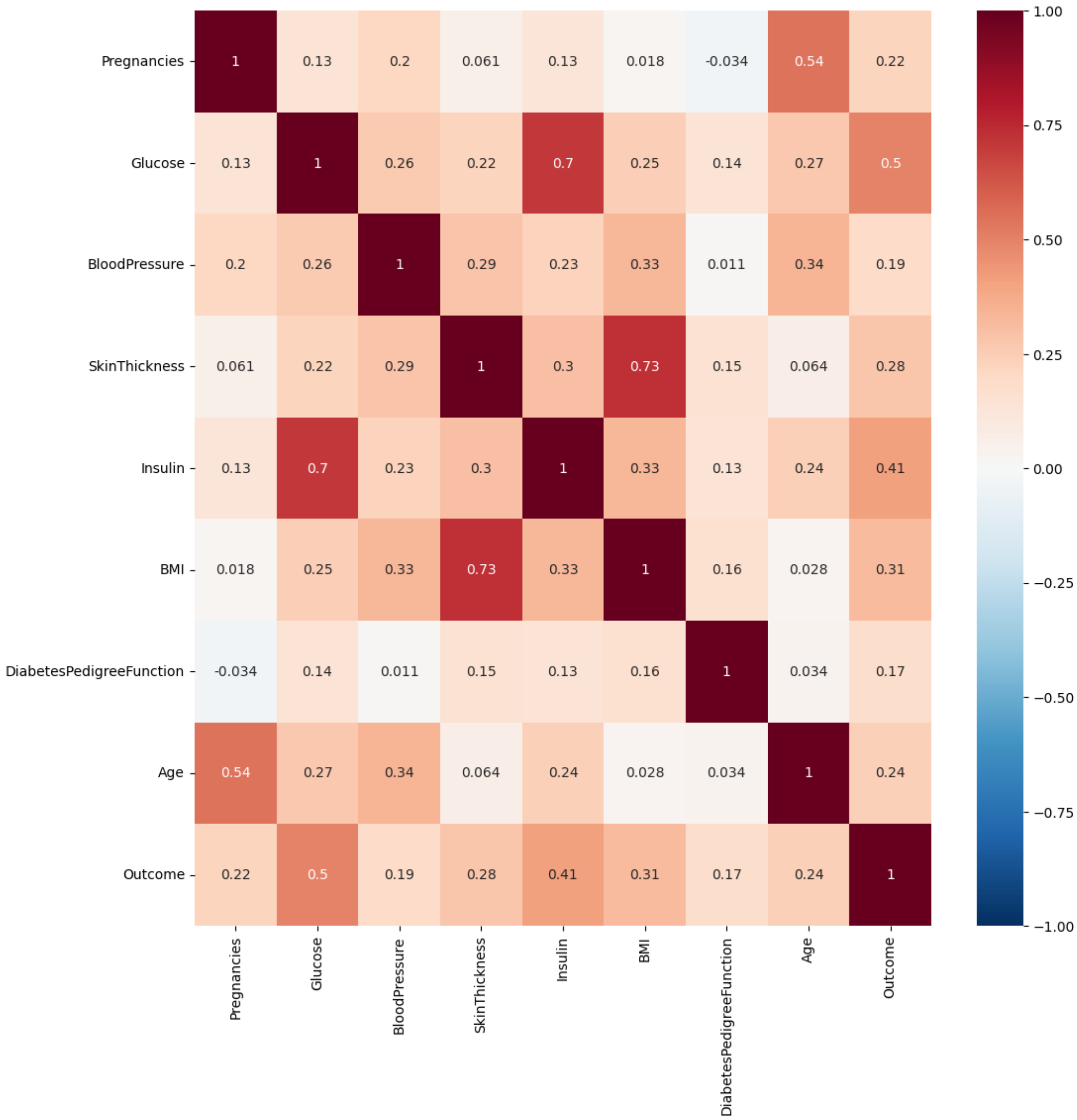
0
1
2
DiabetesPedigreeFunctio





Correlation Analysis Heatmap

[10]:
```
plt.figure(figsize = (12,12))
sns.heatmap(data.corr(), vmin = -1.0, center = 0, cmap = 'RdBu_r', annot = True)
plt.show()
```



Standard Scaling and Label Encoding

[11]:
```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x = pd.DataFrame(sc_x.fit_transform(data.drop(['Outcome'],axis = 1),),
    columns =␣
 ↪['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeF
```

[12]:
```
x.head()
```

[12]:
```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.639947  0.859332      -0.003370       0.481771  0.089650  0.182947
```

```
1    -0.844885 -1.192631      -0.460186       0.010837 -0.844397 -0.822529
2     1.233880  1.999311      -0.612458      -1.245028  2.488832 -1.296539
3    -0.844885 -1.062348      -0.460186      -0.460098 -0.485304 -0.607070
4    -1.141852  0.501052      -2.439721       0.481771  0.021807  1.547521

   DiabetesPedigreeFunction       Age
0                  0.468492  1.425995
1                 -0.365061 -0.190672
2                  0.604397 -0.105584
3                 -0.920763 -1.041549
4                  5.484909 -0.020496
```

[13]:
```python
y = data['Outcome']
```

Test Train Split

[14]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.
 ↪3,random_state = 0)
```

[15]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Lists to hold scores and parameter combinations
train_scores = []
test_scores = []
params_list = []  # To store parameter combinations

# Hyperparameters
distance_metrics = ['euclidean', 'manhattan']
weights_options = ['uniform', 'distance']
```

Find value of K + Hyperparameters

[16]:
```python
 #Loop through different values of k
for k in range(1, 50):
    for metric in distance_metrics:
        for weight in weights_options:
            knn = KNeighborsClassifier(n_neighbors=k, metric=metric,␣
 ↪weights=weight)
            knn.fit(x_train, y_train)

            # Calculate scores
            train_score = knn.score(x_train, y_train)
            test_score = knn.score(x_test, y_test)

            # Append scores and parameters
```

```
            train_scores.append(train_score)
            test_scores.append(test_score)
            params_list.append((k, metric, weight, train_score, test_score))  #␣
 ↪Store train and test scores

# Find the maximum testing score and corresponding parameters
max_test_score = max(test_scores)
test_index = test_scores.index(max_test_score)
best_test_params = params_list[test_index]
```

Result

[17]:
```
 #Output results
print('Max Test score: {:.2f}% at k = {}, metric = {}, weights = {}'.format(
    max_test_score * 100,
    best_test_params[0],
    best_test_params[1],
    best_test_params[2]
))
print('Training score for the best model: {:.2f}%'.format(best_test_params[3] *␣
 ↪100))  # Retrieve training score
print('Testing score for the best model: {:.2f}%'.format(best_test_params[4] *␣
 ↪100))  # Retrieve testing score

# Fit the model with the best parameters
best_knn = KNeighborsClassifier(n_neighbors=best_test_params[0],
                                metric=best_test_params[1],
                                weights=best_test_params[2])
best_knn.fit(x_train, y_train)

# Make predictions
y_pred = best_knn.predict(x_test)

# Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Output confusion matrix and classification report
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

```
Max Test score: 80.09% at k = 30, metric = manhattan, weights = uniform
Training score for the best model: 77.84%
Testing score for the best model: 80.09%

Confusion Matrix:
 [[147  10]
 [ 36  38]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.94      0.86       157
           1       0.79      0.51      0.62        74

    accuracy                           0.80       231
   macro avg       0.80      0.72      0.74       231
weighted avg       0.80      0.80      0.79       231
```

diabetes-prediction-using-decision-tree

October 20, 2024

AIES Mini Project By:-Hrishit Madhavi

[1]:
```
from google.colab import files
import pandas as pd

# Upload the file
uploaded = files.upload()

# Assuming you uploaded 'cleaned_data_logistic_regression.csv'
data = pd.read_csv('cleaned_data_logistic_regression.csv')
```

```
<IPython.core.display.HTML object>
```

```
Saving cleaned_data_logistic_regression.csv to
cleaned_data_logistic_regression.csv
```

Decision Tree Accuracy

[ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Prepare the data (X for features, y for target/labels)
X = data.drop('Outcome', axis=1)  # Drop 'Outcome' column if it's the target
y = data['Outcome']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define the Decision Tree model
model = DecisionTreeClassifier(random_state=42)

# Define the hyperparameter grid
param_grid = {
    'max_depth': [2, 3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 8, 16]
```

```
}

# Use GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
↪scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Predict using the best model
y_pred = best_model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy and best hyperparameters
print(f"Training score for the best model: {grid_search.best_score_ * 100:.
↪2f}%")
print(f"Testing score for the best model: {accuracy * 100:.2f}%")
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Classification Report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

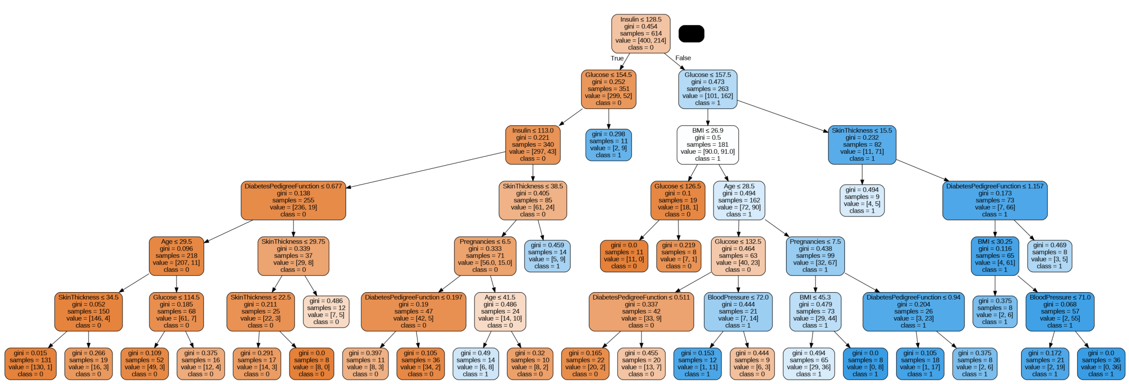
```
Training score for the best model: 80.62%
Testing score for the best model: 71.43%
Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 4,
'min_samples_split': 20}

Confusion Matrix:
[[82 17]
 [27 28]]

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.83      0.79        99
           1       0.62      0.51      0.56        55
```

```
    accuracy                           0.71       154
   macro avg       0.69      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154


/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:
invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Decision Tree Graph

[ ]:
```python
# prompt: make decision tree graph

from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

# Assuming 'best_model' is your trained Decision Tree model

dot_data = StringIO()
export_graphviz(best_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names=X.columns,  # Replace with your feature names
                class_names=['0', '1'])  # Replace with your class names

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

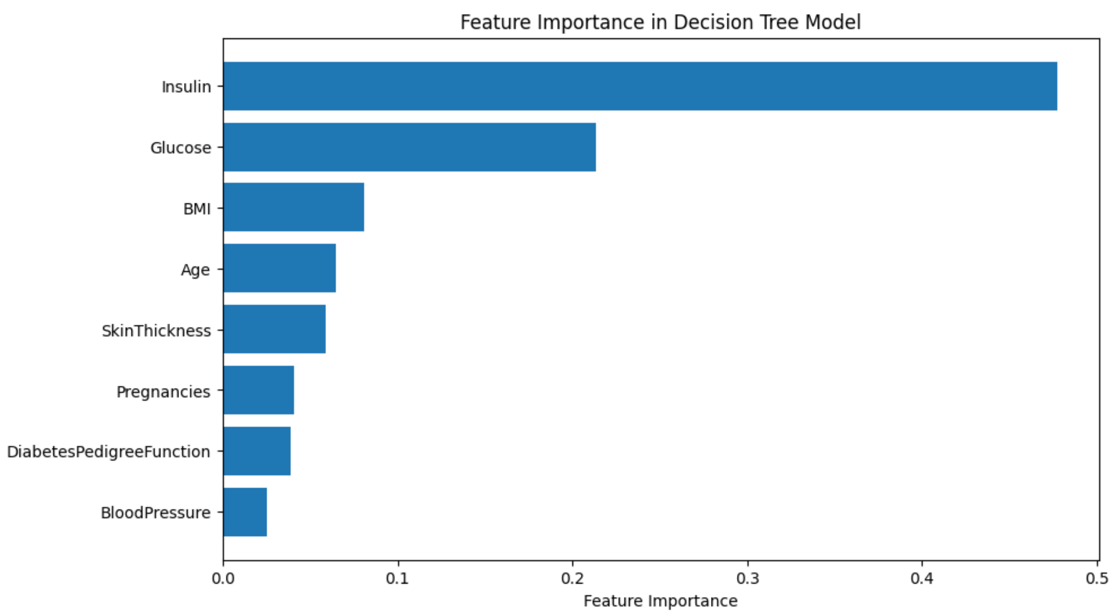
[ ]:



Feature Selection in decision tree

[ ]:
```python
# Assuming 'best_model' is your trained Decision Tree model
dot_data = StringIO()
export_graphviz(best_model, out_file=dot_data,
                filled=True, rounded=True,
```

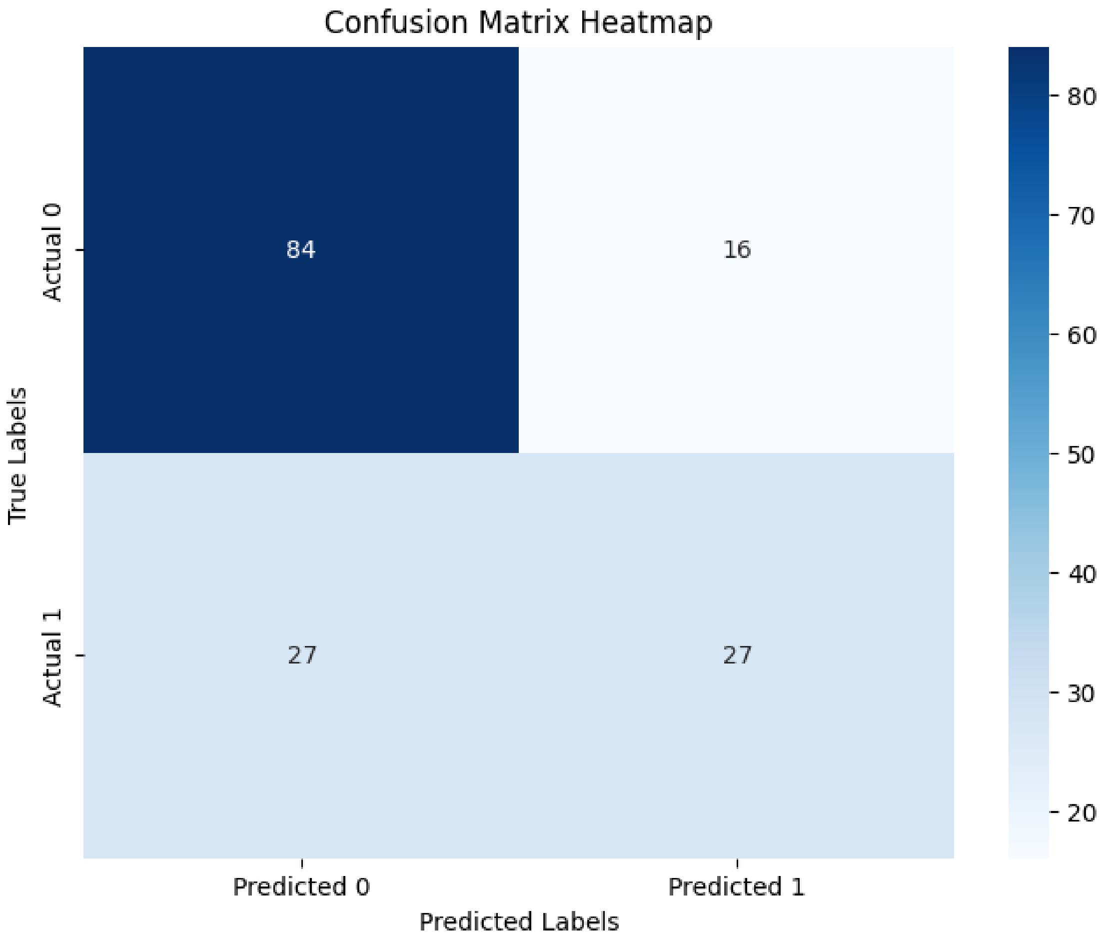
```
                special_characters=True,
                feature_names=X.columns,
                class_names=['0', '1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())

# Feature Importance plot
import matplotlib.pyplot as plt

feature_importances = best_model.feature_importances_
sorted_idx = feature_importances.argsort()

plt.figure(figsize=(10, 6))
plt.barh(X.columns[sorted_idx], feature_importances[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Feature Importance in Decision Tree Model")
plt.show()

# Tree depth
tree_depth = best_model.tree_.max_depth
print(f"Tree Depth: {tree_depth}")
```




```
Tree Depth: 6
```

Confusion Matrix

[10]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is already loaded and prepared
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define and train the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Create the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```



Decision Tree Accuracy Increased

```
[ ]: from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Prepare the data
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Split the data (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Define the model
model = DecisionTreeClassifier(random_state=42)
```

```python
# Hyperparameter grid
param_grid = {
    'max_depth': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'criterion': ['gini', 'entropy']
}

# GridSearchCV for tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model and evaluate
best_model = grid_search.best_estimator_
train_accuracy = best_model.score(X_train, y_train)
y_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Print accuracies and best hyperparameters
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Confusion Matrix and Classification Report
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

```
Training Accuracy: 84.36%
Testing Accuracy: 79.87%
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf':
8, 'min_samples_split': 2}

Confusion Matrix:
[[83 17]
 [14 40]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.83      0.84       100
           1       0.70      0.74      0.72        54
```

```
    accuracy                           0.80       154
   macro avg       0.78      0.79      0.78       154
weighted avg       0.80      0.80      0.80       154
```

prediction-using-random-forest-v1

October 21, 2024

[ ]:
```
from google.colab import files
uploaded= files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving diabetes.csv to diabetes.csv
```

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve,␣
 ↪classification_report
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('diabetes.csv')
df.head()
```

[ ]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Diabetes Pridection Using Random Forest

Rondom Forest

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating a number of Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.

[ ]:
```
df.info()
```

```
<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
```

1 Checking for missing values

[ ]:
```
df.isnull().sum()
```

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

2 Mapping the 'Outcome' column to more descriptive categories

[ ]:
```
df['Outcome']=np.where(df['Outcome']==1,'Diabetic','No Diabetic')
df.head()
```

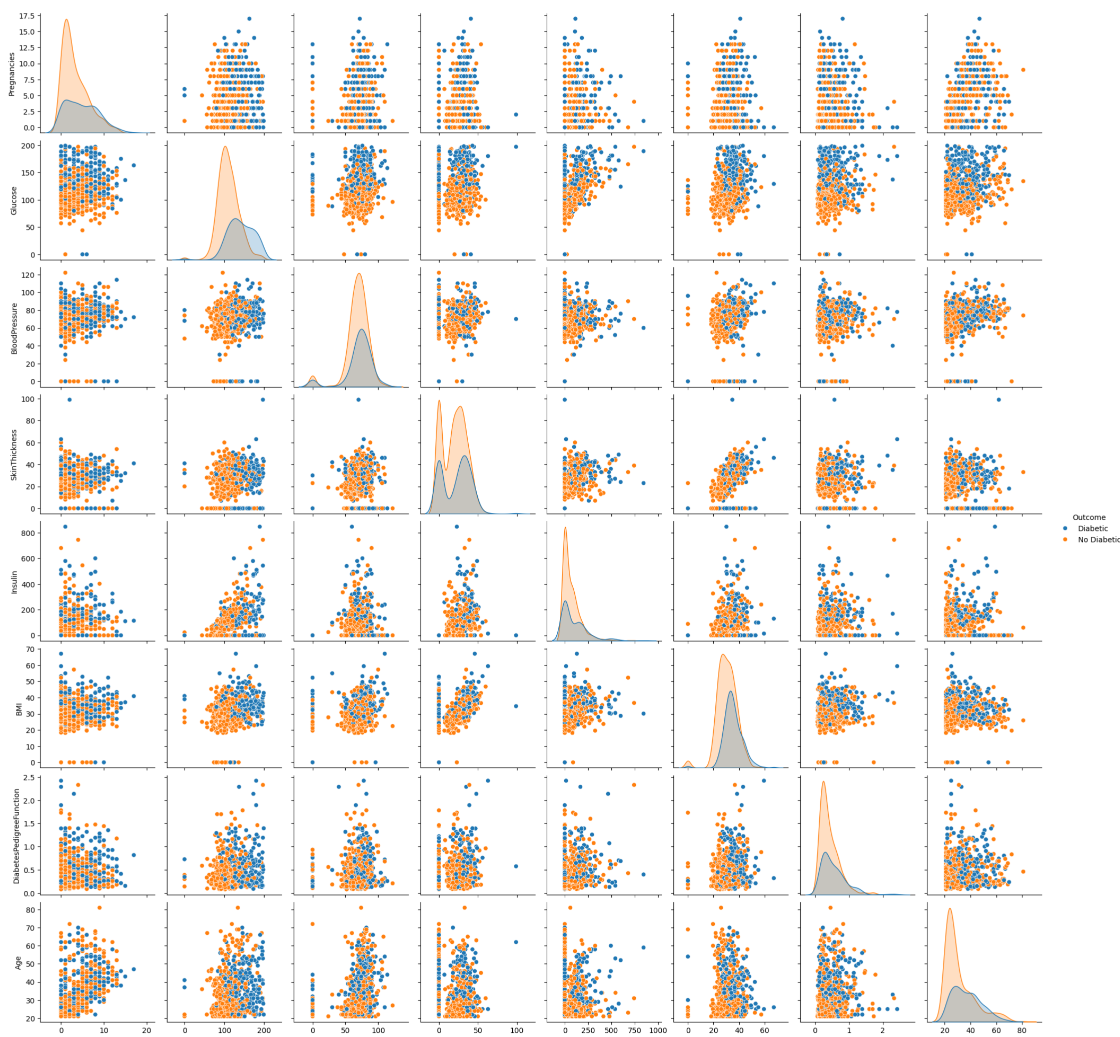
[ ]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
```

```
2            8     183         64          0        0  23.3
3            1      89         66         23       94  28.1
4            0     137         40         35      168  43.1

   DiabetesPedigreeFunction  Age      Outcome
0                     0.627   50     Diabetic
1                     0.351   31  No Diabetic
2                     0.672   32     Diabetic
3                     0.167   21  No Diabetic
4                     2.288   33     Diabetic
```

3 Visualizing pairwise relationships in the dataset

[ ]:
```python
sns.pairplot(df,hue="Outcome")
```

[ ]: <seaborn.axisgrid.PairGrid at 0x7dbc84495e70>



[ ]:
```
df.describe()
```

[ ]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age
count  768.000000                768.000000  768.000000
mean    31.992578                  0.471876   33.240885
std      7.884160                  0.331329   11.760232
min      0.000000                  0.078000   21.000000
25%     27.300000                  0.243750   24.000000
50%     32.000000                  0.372500   29.000000
75%     36.600000                  0.626250   41.000000
max     67.100000                  2.420000   81.000000
```

[ ]:
```
df = pd.read_csv('diabetes.csv')
df.head()
```

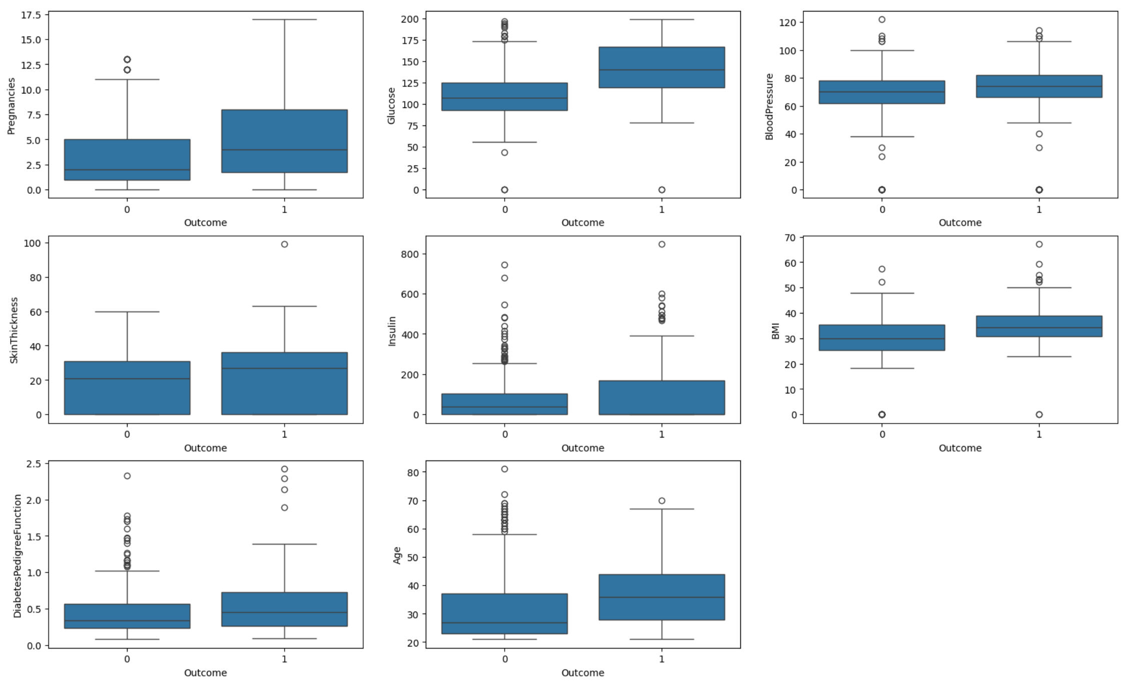
[ ]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

4 Checking the distribution of the target variable

[ ]:
```
df['Outcome'].value_counts()
```

[ ]: Outcome
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

[ ]:
```
#Visualizing numerical variables

plt.figure(figsize=(20, 12))
plt.subplot(3,3,1)
sns.boxplot(x = 'Outcome', y = 'Pregnancies', data = df)
plt.subplot(3,3,2)
sns.boxplot(x = 'Outcome', y = 'Glucose', data = df)
plt.subplot(3,3,3)
sns.boxplot(x = 'Outcome', y = 'BloodPressure', data = df)
plt.subplot(3,3,4)
sns.boxplot(x = 'Outcome', y = 'SkinThickness', data = df)
plt.subplot(3,3,5)
sns.boxplot(x = 'Outcome', y = 'Insulin', data = df)
plt.subplot(3,3,6)
sns.boxplot(x = 'Outcome', y = 'BMI', data = df)
plt.subplot(3,3,7)
sns.boxplot(x = 'Outcome', y = 'DiabetesPedigreeFunction', data = df)
plt.subplot(3,3,8)
sns.boxplot(x = 'Outcome', y = 'Age', data = df)
plt.show()
```



Train Test Split the Data

Function to split the dataset into training and testing sets

```
[ ]: def train_test_split_and_features(df):
         y = df["Outcome"]
         # Remove 'Id' from the list of columns to drop if it doesn't exist.
         x = df.drop(['Outcome'],axis=1)
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,␣
     ↪random_state = 0)
         print(x.head(5))
         print(x.columns)
         features = list(x.columns)
         return x_train, x_test, y_train, y_test,features
```

Splitting the dataset

```
[ ]: x_train, x_test, y_train, y_test,features = train_test_split_and_features(df)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

Function to fit the model and evaluate its performance

```
[ ]: def fit_and_evaluate_model(x_train, x_test, y_train,␣
     ↪y_test,max_depth=5,min_samples_split=0.01,max_features=0.8,max_samples=0.8):
         random_forest =  RandomForestClassifier(random_state=0,\
                                                 max_depth=max_depth,\
                                                ␣
     ↪min_samples_split=min_samples_split,\
                                                 max_features=max_features,
                                                 max_samples=max_samples)

         model = random_forest.fit(x_train, y_train)
         random_forest_predict = random_forest.predict(x_test)
```

```
    random_forest_conf_matrix = confusion_matrix(y_test, random_forest_predict)
    random_forest_acc_score = accuracy_score(y_test, random_forest_predict)
    print("confussion matrix")
    print(random_forest_conf_matrix)
    print("\n")
    print("Accuracy of Random Forest:",random_forest_acc_score*100,'\n')
    print(classification_report(y_test,random_forest_predict))
    return model
```

Fitting the model and evaluating its performance

[ ]:
```
model = fit_and_evaluate_model(x_train, x_test, y_train, y_test)
```

```
confussion matrix
[[95 12]
 [19 28]]



Accuracy of Random Forest: 79.87012987012987

              precision    recall  f1-score   support

           0       0.83      0.89      0.86       107
           1       0.70      0.60      0.64        47

    accuracy                           0.80       154
   macro avg       0.77      0.74      0.75       154
weighted avg       0.79      0.80      0.79       154
```

Hyperparameter tuning using GridSearchCV

[ ]:
```
param_grid = [
  {'max_depth': [3,5,7,10], 'min_samples_split': [0.01,0.03, 0.07, 0.1],
'max_features': [0.7,0.8,0.9,1.0],
'max_samples': [0.7,0.8,0.9,1.0]}]
```

[ ]:
```
model = RandomForestClassifier()
search = GridSearchCV(estimator = model, param_grid = param_grid, cv=5,␣
 ↪verbose=5)
search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 256 candidates, totalling 1280 fits
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.01;, score=0.715 total time=   0.2s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.01;, score=0.805 total time=   0.2s
[CV 3/5] END max_depth=3, max_features=0.7, max_samples=0.7,
```

```
min_samples_split=0.01;, score=0.780 total time=   0.2s
[CV 4/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.01;, score=0.707 total time=   0.2s
[CV 5/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.01;, score=0.770 total time=   0.2s
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.03;, score=0.732 total time=   0.2s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.03;, score=0.829 total time=   0.2s
[CV 3/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.03;, score=0.780 total time=   0.2s
[CV 4/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.03;, score=0.707 total time=   0.2s
[CV 5/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.03;, score=0.770 total time=   0.2s
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.07;, score=0.732 total time=   0.2s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.07;, score=0.813 total time=   0.2s
[CV 3/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.07;, score=0.789 total time=   0.2s
[CV 4/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.07;, score=0.699 total time=   0.2s
[CV 5/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.07;, score=0.779 total time=   0.2s
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.1;, score=0.715 total time=   0.2s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.1;, score=0.821 total time=   0.2s
[CV 3/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.1;, score=0.764 total time=   0.2s
[CV 4/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.1;, score=0.707 total time=   0.3s
[CV 5/5] END max_depth=3, max_features=0.7, max_samples=0.7,
min_samples_split=0.1;, score=0.779 total time=   0.3s
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.01;, score=0.707 total time=   0.3s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.01;, score=0.805 total time=   0.3s
[CV 3/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.01;, score=0.772 total time=   0.3s
[CV 4/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.01;, score=0.732 total time=   0.3s
[CV 5/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.01;, score=0.779 total time=   0.3s
[CV 1/5] END max_depth=3, max_features=0.7, max_samples=0.8,
min_samples_split=0.03;, score=0.732 total time=   0.3s
[CV 2/5] END max_depth=3, max_features=0.7, max_samples=0.8,
```

```
min_samples_split=0.07;, score=0.762 total time=   0.3s
[CV 1/5] END max_depth=10, max_features=1.0, max_samples=1.0,
min_samples_split=0.1;, score=0.732 total time=   0.3s
[CV 2/5] END max_depth=10, max_features=1.0, max_samples=1.0,
min_samples_split=0.1;, score=0.789 total time=   0.3s
[CV 3/5] END max_depth=10, max_features=1.0, max_samples=1.0,
min_samples_split=0.1;, score=0.780 total time=   0.3s
[CV 4/5] END max_depth=10, max_features=1.0, max_samples=1.0,
min_samples_split=0.1;, score=0.707 total time=   0.3s
[CV 5/5] END max_depth=10, max_features=1.0, max_samples=1.0,
min_samples_split=0.1;, score=0.787 total time=   0.3s
```

[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid=[{'max_depth': [3, 5, 7, 10],
                          'max_features': [0.7, 0.8, 0.9, 1.0],
                          'max_samples': [0.7, 0.8, 0.9, 1.0],
                          'min_samples_split': [0.01, 0.03, 0.07, 0.1]}],
             verbose=5)
```

[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid=[{'max_depth': [3, 5, 7, 10],
                          'max_features': [0.7, 0.8, 0.9, 1.0],
                          'max_samples': [0.7, 0.8, 0.9, 1.0],
                          'min_samples_split': [0.01, 0.03, 0.07, 0.1]}],
             verbose=5)
```

[ ]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid=[{'max_depth': [3, 5, 7, 10],
                          'max_features': [0.7, 0.8, 0.9, 1.0],
                          'max_samples': [0.7, 0.8, 0.9, 1.0],
                          'min_samples_split': [0.01, 0.03, 0.07, 0.1]}],
             verbose=5)
```

Displaying the grid search results

[ ]:
```
results = pd.DataFrame(search.cv_results_)
results.sort_values('mean_test_score',inplace=True,ascending= False)
results.head(10)
```

[ ]:
```
     mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
107       0.230993      0.007293         0.009527        0.000617
211       0.218361      0.005778         0.010823        0.000779
70        0.311702      0.003493         0.013147        0.000738
76        0.226881      0.006853         0.010279        0.000983
111       0.363710      0.007710         0.015146        0.001296
103       0.225562      0.007187         0.012770        0.003096
167       0.226354      0.004029         0.009977        0.001166
69        0.252926      0.046225         0.011375        0.002133
```

```
215       0.288611      0.062354         0.013709        0.003614
139       0.267975      0.054308         0.011094        0.001670

    param_max_depth param_max_features param_max_samples  \
107               5                0.9               0.9
211              10                0.8               0.7
70                5                0.7               0.8
76                5                0.7               1.0
111               5                0.9               1.0
103               5                0.9               0.8
167               7                0.9               0.8
69                5                0.7               0.8
215              10                0.8               0.8
139               7                0.7               0.9

    param_min_samples_split  \
107                     0.1
211                     0.1
70                     0.07
76                     0.01
111                     0.1
103                     0.1
167                     0.1
69                     0.03
215                     0.1
139                     0.1

                                                params  split0_test_score  \
107  {'max_depth': 5, 'max_features': 0.9, 'max_sam…           0.715447
211  {'max_depth': 10, 'max_features': 0.8, 'max_sa…           0.715447
70   {'max_depth': 5, 'max_features': 0.7, 'max_sam…           0.715447
76   {'max_depth': 5, 'max_features': 0.7, 'max_sam…           0.715447
111  {'max_depth': 5, 'max_features': 0.9, 'max_sam…           0.723577
103  {'max_depth': 5, 'max_features': 0.9, 'max_sam…           0.747967
167  {'max_depth': 7, 'max_features': 0.9, 'max_sam…           0.723577
69   {'max_depth': 5, 'max_features': 0.7, 'max_sam…           0.723577
215  {'max_depth': 10, 'max_features': 0.8, 'max_sa…           0.707317
139  {'max_depth': 7, 'max_features': 0.7, 'max_sam…           0.723577

     split1_test_score  split2_test_score  split3_test_score  \
107           0.804878           0.780488           0.731707
211           0.821138           0.796748           0.715447
70            0.804878           0.780488           0.731707
76            0.788618           0.804878           0.731707
111           0.813008           0.780488           0.731707
103           0.804878           0.788618           0.715447
167           0.788618           0.804878           0.699187
```

```
69              0.813008            0.788618            0.691057
215             0.804878            0.796748            0.715447
139             0.804878            0.788618            0.715447

     split4_test_score  mean_test_score  std_test_score  rank_test_score
107           0.803279         0.767160        0.036976                1
211           0.786885         0.767133        0.043650                2
70            0.795082         0.765520        0.035490                3
76            0.786885         0.765507        0.035184                4
111           0.778689         0.765494        0.033332                5
103           0.770492         0.765480        0.031377                6
167           0.803279         0.763908        0.043942                7
69            0.803279         0.763908        0.047969                7
215           0.795082         0.763894        0.043081                9
139           0.786885         0.763881        0.036857               10
```

[ ]:
```
results_save = pd.DataFrame(search.cv_results_)
results_save.to_csv("results_save.csv", index =False)
```

[ ]:
```
search.best_params_
```

[ ]:
```
{'max_depth': 5,
 'max_features': 0.9,
 'max_samples': 0.9,
 'min_samples_split': 0.1}
```

Re-evaluating the model with best parameters

[ ]:
```
model = fit_and_evaluate_model(x_train, x_test, y_train, y_test,␣
 ↪max_depth=10,min_samples_split=0.01,\
                               max_features=0.7,max_samples= 1.0)
```

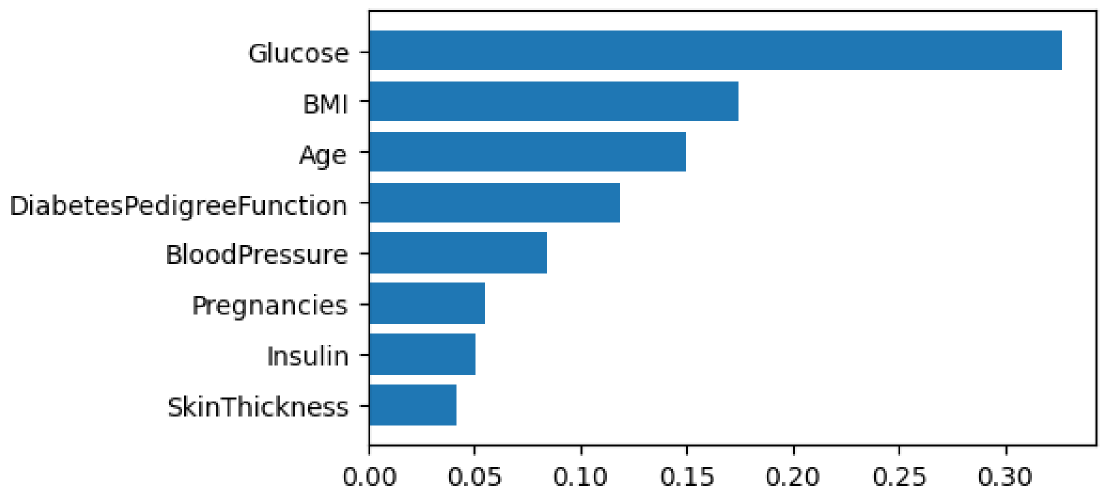
```
confussion matrix
[[92 15]
 [13 34]]


Accuracy of Random Forest: 81.81818181818183

              precision    recall  f1-score   support

           0       0.88      0.86      0.87       107
           1       0.69      0.72      0.71        47

    accuracy                           0.82       154
   macro avg       0.79      0.79      0.79       154
weighted avg       0.82      0.82      0.82       154
```

[ ]:
```
importances = pd.DataFrame(model.feature_importances_)
importances['features'] = features
importances.columns = ['importance','feature']
importances.sort_values(by = 'importance', ascending= True,inplace=True)
```

[ ]:
```
# Feature importances
```

[ ]:
```
plt.figure(figsize=(5, 3))
plt.barh(importances.feature, importances.importance)
```

[ ]: <BarContainer object of 8 artists>



[ ]:
```
!pip install gradio
```

```
Collecting gradio
  Downloading gradio-4.44.0-py3-none-any.whl.metadata (15 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0 (from gradio)
  Downloading fastapi-0.115.0-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.4.0-py3-none-any.whl.metadata (2.9 kB)
Collecting gradio-client==1.3.0 (from gradio)
  Downloading gradio_client-1.3.0-py3-none-any.whl.metadata (7.1 kB)
Collecting httpx>=0.24.1 (from gradio)
  Downloading httpx-0.27.2-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: huggingface-hub>=0.19.3 in
/usr/local/lib/python3.10/dist-packages (from gradio) (0.24.7)
```

```
Requirement already satisfied: importlib-resources<7.0,>=1.3 in
/usr/local/lib/python3.10/dist-packages (from gradio) (6.4.5)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.10/dist-
packages (from gradio) (3.1.4)
Requirement already satisfied: markupsafe~=2.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (2.1.5)
Requirement already satisfied: matplotlib~=3.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (3.7.1)
Requirement already satisfied: numpy<3.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (1.26.4)
Collecting orjson~=3.0 (from gradio)
  Downloading orjson-3.10.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_
64.whl.metadata (50 kB)
                           50.4/50.4 kB
2.8 MB/s eta 0:00:00
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from gradio) (24.1)
Requirement already satisfied: pandas<3.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (2.1.4)
Requirement already satisfied: pillow<11.0,>=8.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (10.4.0)
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.10/dist-
packages (from gradio) (2.9.2)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.9 (from gradio)
  Downloading python_multipart-0.0.10-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.2.2 (from gradio)
  Downloading ruff-0.6.7-py3-none-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting tomlkit==0.12.0 (from gradio)
  Downloading tomlkit-0.12.0-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in
/usr/local/lib/python3.10/dist-packages (from gradio) (0.12.5)
Requirement already satisfied: typing-extensions~=4.0 in
/usr/local/lib/python3.10/dist-packages (from gradio) (4.12.2)
Requirement already satisfied: urllib3~=2.0 in /usr/local/lib/python3.10/dist-
packages (from gradio) (2.0.7)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.30.6-py3-none-any.whl.metadata (6.6 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from gradio-client==1.3.0->gradio) (2024.6.1)
Collecting websockets<13.0,>=10.0 (from gradio-client==1.3.0->gradio)
  Downloading websockets-12.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64
```

```
.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.6 kB)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-
packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-
packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-
packages (from anyio<5.0,>=3.0->gradio) (1.2.2)
Collecting starlette<0.39.0,>=0.37.2 (from fastapi<1.0->gradio)
  Downloading starlette-0.38.6-py3-none-any.whl.metadata (6.0 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx>=0.24.1->gradio) (2024.8.30)
Collecting httpcore==1.* (from httpx>=0.24.1->gradio)
  Downloading httpcore-1.0.5-py3-none-any.whl.metadata (20 kB)
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx>=0.24.1->gradio)
  Downloading h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.19.3->gradio) (3.16.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.19.3->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub>=0.19.3->gradio) (4.66.5)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib~=3.0->gradio) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas<3.0,>=1.0->gradio) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas<3.0,>=1.0->gradio) (2024.1)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.0->gradio) (2.23.4)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-
packages (from typer<1.0,>=0.12->gradio) (8.1.7)
Requirement already satisfied: shellingham>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/dist-
packages (from typer<1.0,>=0.12->gradio) (13.8.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
```

```
packages (from python-dateutil>=2.7->matplotlib~=3.0->gradio) (1.16.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from
rich>=10.11.0->typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from
rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.18.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-
hub>=0.19.3->gradio) (3.3.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio)
(0.1.2)
Downloading gradio-4.44.0-py3-none-any.whl (18.1 MB)
                         18.1/18.1 MB
51.7 MB/s eta 0:00:00
Downloading gradio_client-1.3.0-py3-none-any.whl (318 kB)
                         318.7/318.7 kB
20.4 MB/s eta 0:00:00
Downloading tomlkit-0.12.0-py3-none-any.whl (37 kB)
Downloading aiofiles-23.2.1-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.0-py3-none-any.whl (94 kB)
                         94.6/94.6 kB
6.4 MB/s eta 0:00:00
Downloading httpx-0.27.2-py3-none-any.whl (76 kB)
                         76.4/76.4 kB
5.2 MB/s eta 0:00:00
Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
                         77.9/77.9 kB
5.5 MB/s eta 0:00:00
Downloading
orjson-3.10.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (141
kB)
                         141.9/141.9 kB
8.6 MB/s eta 0:00:00
Downloading python_multipart-0.0.10-py3-none-any.whl (22 kB)
Downloading ruff-0.6.7-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(10.8 MB)
                         10.8/10.8 MB
62.9 MB/s eta 0:00:00
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading uvicorn-0.30.6-py3-none-any.whl (62 kB)
                         62.8/62.8 kB
4.5 MB/s eta 0:00:00
Downloading ffmpy-0.4.0-py3-none-any.whl (5.8 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Downloading h11-0.14.0-py3-none-any.whl (58 kB)
                         58.3/58.3 kB
```

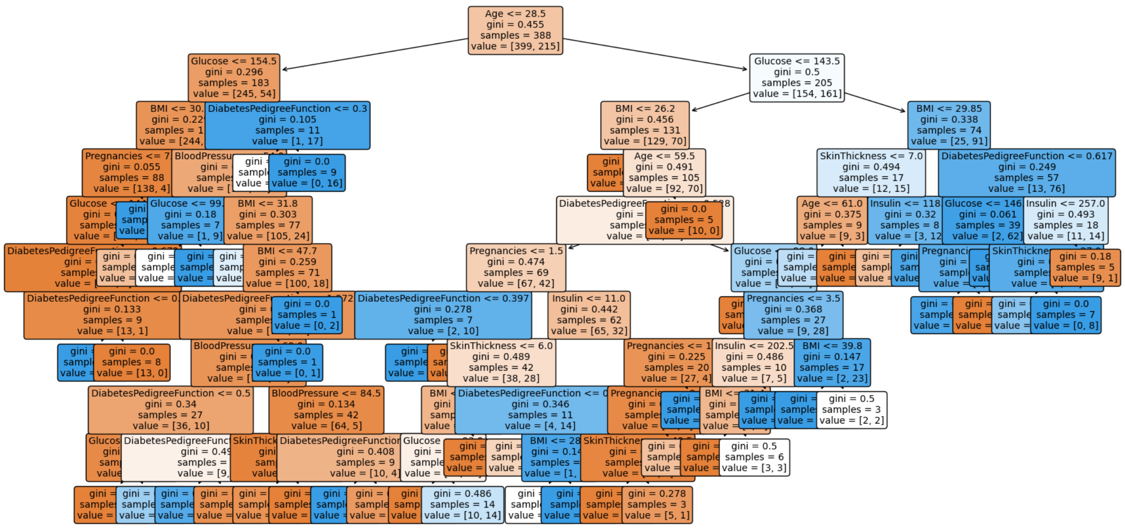
```
4.3 MB/s eta 0:00:00
Downloading starlette-0.38.6-py3-none-any.whl (71 kB)
                         71.5/71.5 kB
5.2 MB/s eta 0:00:00
Downloading websockets-12.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x8
6_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (130 kB)
                         130.2/130.2 kB
9.5 MB/s eta 0:00:00
Installing collected packages: pydub, websockets, tomlkit, semantic-
version, ruff, python-multipart, orjson, h11, ffmpy, aiofiles, uvicorn,
starlette, httpcore, httpx, fastapi, gradio-client, gradio
Successfully installed aiofiles-23.2.1 fastapi-0.115.0 ffmpy-0.4.0 gradio-4.44.0
gradio-client-1.3.0 h11-0.14.0 httpcore-1.0.5 httpx-0.27.2 orjson-3.10.7
pydub-0.25.1 python-multipart-0.0.10 ruff-0.6.7 semantic-version-2.10.0
starlette-0.38.6 tomlkit-0.12.0 uvicorn-0.30.6 websockets-12.0
```

[ ]:
```
!apt-get install graphviz
!pip install pydotplus
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
graphviz is already the newest version (2.42.2-6ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
Requirement already satisfied: pydotplus in /usr/local/lib/python3.10/dist-
packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in
/usr/local/lib/python3.10/dist-packages (from pydotplus) (3.1.4)
```

[ ]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import graphviz
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
```

[ ]:
```
# Select one tree from the forest
estimator = model.estimators_[0]  # The first tree in the Random Forest

# Visualizing the tree structure
plt.figure(figsize=(20, 10))
plot_tree(estimator, feature_names=features, filled=True, rounded=True,
 ↪fontsize=10)
plt.show()
```



[ ]:
```python
dot_data = export_graphviz(estimator, out_file=None,
                           feature_names=features,
                           class_names=['No Diabetic', 'Diabetic'],
                           filled=True, rounded=True,
                           special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

[ ]: