Assignment Title: Classifier Evaluation

Introduction: In this assignment, you will learn how to evaluate the performance of a classifier using various evaluation metrics. You will be provided with a dataset and a pretrained classifier. Your task is to evaluate the classifier's performance using the metrics discussed in class and answer the following questions.

Dataset: You will be provided with a dataset containing information about patients with diabetes. The dataset consists of 768 samples, with each sample containing 8 features. The target variable is a binary variable indicating whether the patient has diabetes or not.

Classifier: Develop a Decision tree and Random forest classifier.

Classifier: Develop a Decision tree and Random forest classifier. Instructions:

1. Load the dataset and the develop a classifier.
2. Split the dataset into training and testing sets using a 70:30 split.
3. Use the classifier to predict the target variable for the testing set.
4. Calculate the following evaluation metrics for the classifier: accuracy, precision, recall, f1-score, and confusion matrix.
5. Answer the following questions:

Questions:

1. What is the accuracy of the classifier on the testing set?
2. What is the precision of the classifier on the testing set?
3. What is the recall of the classifier on the testing set?
4. What is the f1-score of the classifier on the testing set?
5. Interpret the confusion matrix for the classifier on the testing set

STEP 1: IMPORT LIBRARIES

1. Pandas: a library for data manipulation and analysis.
2. NumPy: a library for scientific computing in Python.
3. Scikit-learn: a machine learning library for Python that provides a variety of tools for data analysis and modeling, including train_test_split for splitting data, DecisionTreeClassifier for creating a decision tree classifier, RandomForestClassifier for creating a random forest classifier, and metrics for evaluating model performance.
4. Matplotlib: a library for creating static, animated, and interactive visualizations in Python.
5. Pandas Profiling: a library for generating data analysis reports in HTML format.
6. IPython: an interactive shell for Python.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from IPython.core.display import display, HTML
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import pandas_profiling
```

STEP 2: IMPORT DATASET

The code loads the diabetes dataset from a CSV file named "diabetes.csv" and stores it as a Pandas DataFrame named "df".

In [2]:
```
# Load the breast cancer dataset
df = pd.read_csv('diabetes.csv')
```

The code creates a Pandas Profiling report for the "df" DataFrame using the ProfileReport function from the pandas_profiling library. The title parameter specifies the title of the report, and the explorative parameter specifies whether to perform an exploratory data analysis or not. The resulting report will contain various statistics and visualizations to help understand the data better.

In [3]:
```
profile = pandas_profiling.ProfileReport(df, title="Analysis Report", explorative=True)
```

In [4]:
```
profile
```

Overview

Dataset statistics

Number of variables	9
Number of observations	768
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	54.1 KiB
Average record size in memory	72.2 B

Variable types

Numeric	8
Categorical	1

Alerts

`Pregnancies` is highly correlated with `Age`	**High correlation**
`SkinThickness` is highly correlated with `Insulin`	**High correlation**
`Insulin` is highly correlated with `SkinThickness`	**High correlation**
`Age` is highly correlated with `Pregnancies`	**High correlation**
`Pregnancies` is highly correlated with `Age`	**High correlation**
`Age` is highly correlated with `Pregnancies`	**High correlation**
`Pregnancies` is highly correlated with `Age`	**High correlation**

Out[4]:

STEP 3: CREATE X (features) and y (target)

The code creates a new DataFrame named "X" by dropping the column named "Outcome" from the "df" DataFrame. The drop method in Pandas is used to remove one or more columns from a DataFrame, and the axis parameter is set to 1 to indicate that we are dropping a column (0 would indicate that we are dropping a row). The resulting DataFrame "X" will contain all the columns in "df" except the "Outcome" column.

In [5]:
```
X = df.drop(['Outcome'], axis=1)
```

The code creates a new Series named "y" by selecting the "Outcome" column from the "df" DataFrame. In this case, "Outcome" is the target variable, and we are storing it separately from the input features. The resulting Series "y" will contain the target variable values for each observation in the dataset.

In [6]:
```
y = df['Outcome']
```

STEP 4: PARTITION OF DATASET

The code splits the data into training and test sets using the train_test_split function from scikit-learn. The input DataFrame "X" and Series "y" are split into four separate arrays: "X_train" (the input features for the training set), "X_test" (the input features for the test set), "y_train" (the target variable values for the training set), and "y_test" (the target variable values for the test set). The test_size parameter is set to 0.3, which means that 30% of the data will be used for testing, and the remaining 70% will be used for training. The random_state parameter is set to 42, which is a fixed seed value used to ensure reproducibility of the results.

In [7]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

STEP 5: CREATE A RANDOM FOREST CLASSIFIER WITH 50 TREES

The code creates a Random Forest Classifier model (rfc) with 50 estimators using the RandomForestClassifier class from scikit-learn. The n_estimators parameter specifies the number of trees in the forest.

After creating the model, the code fits the model to the training data using the fit method, with X_train and y_train as arguments. This means that the model will learn to predict the target variable (y) from the input features (X) based on the patterns in the training data.

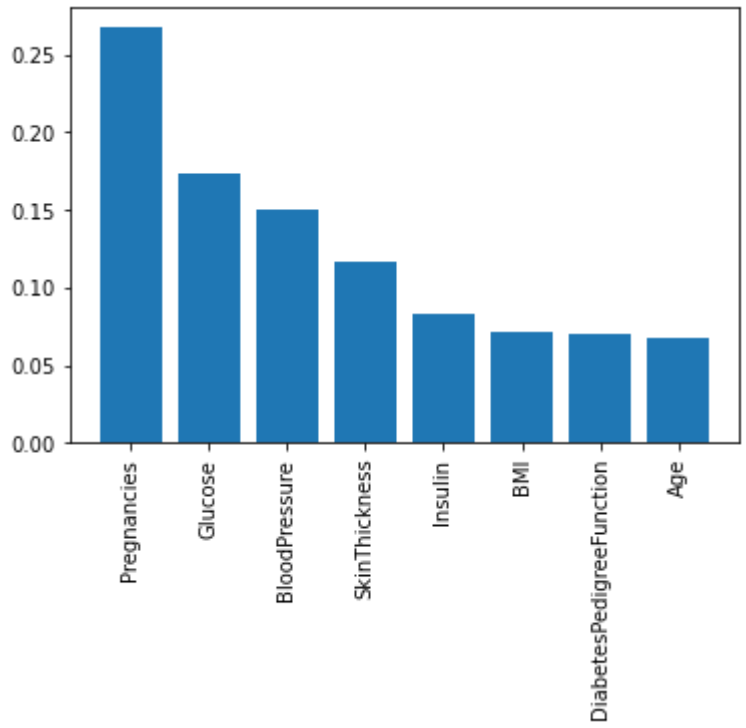
In [8]:
```
rfc = RandomForestClassifier(n_estimators=50)

# Fit the model to the training data
rfc.fit(X_train, y_train)
```

Out[8]:
```
RandomForestClassifier(n_estimators=50)
```

STEP 6: PLOT FEATURE IMPORTANCE

The code calculates the feature importances for the Random Forest Classifier model (rfc) and stores them in the importances array. The indices of the features are then sorted based on their importance score in descending order, and the resulting order is stored in the indices array. The feature importances are then visualized using a bar chart created with Matplotlib's bar function. The x-axis values of the chart are the feature indices, and the y-axis values are the importance scores. The xticks function is used to set the tick labels on the x-axis to the names of the features in the original DataFrame "X". Finally, the show function is used to display the bar chart.

In [9]:
```
# Get the feature importances and plot them
importances = rfc.feature_importances_
indices = sorted(range(len(importances)), key=lambda k: importances[k], reverse=True)
plt.bar(range(X_train.shape[1]), importances[indices])
plt.xticks(range(X_train.shape[1]), X.columns, rotation=90)
plt.show()
```



The code makes predictions on the test data using the predict method of the Random Forest Classifier model (rfc) and stores the predictions in the y_pred variable.

Then, the code prints the accuracy score of the model on the test data. The score method of the model is used to calculate the accuracy score, which is the percentage of correctly classified samples in the test set. The function print is used to display the accuracy score on the console.

In [10]:
```
# Make predictions on the test data
y_pred = rfc.predict(X_test)

# Print the accuracy score of the model on the test data
print("Accuracy:", rfc.score(X_test, y_test))
```

```
Accuracy: 0.7445887445887446
```

STEP 7: CREATE A DECISION TREE CLASSIFIER

The code creates a Decision Tree Classifier model (dtree) using the DecisionTreeClassifier class from scikit-learn. By default, this model will create a tree that partitions the feature space recursively into subsets that are as homogeneous as possible with respect to the target variable. The goal of the model is to learn to predict the target variable from the input features based on the patterns in the training data.

In [11]:
```
dtree = DecisionTreeClassifier()
```

The code fits the Decision Tree Classifier model (dtree) to the training data using the fit method with X_train and y_train as arguments. This means that the model will learn to predict the target variable (y) from the input features (X) based on the patterns in the training data. The fit method adjusts the parameters of the model to minimize the difference between the predicted and actual values of the target variable on the training data.

In [12]:
```
dtree = dtree.fit(X_train, y_train)
```

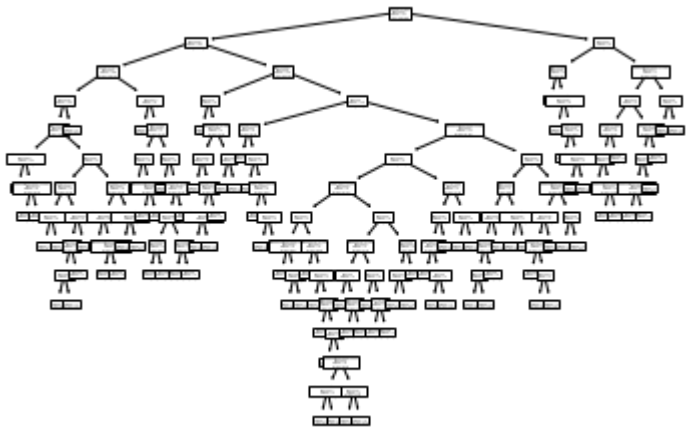
The tree.plot_tree function from scikit-learn's tree module is used to visualize the decision tree model (dtree) that was trained earlier. The function takes the trained model and the names of the features (X.columns) as inputs, and produces a graphical representation of the decision tree. The tree shows how the model makes decisions at each node based on the input features, leading to the final prediction at the leaf nodes. The size of the nodes and the color of the branches can be used to indicate the number of samples that passed through that node during training, and the proportion of samples that belong to each class in the case of a classification problem. This visualization can help to understand the decision-making process of the model and identify potential areas for improvement or overfitting.

In [13]:
```
tree.plot_tree(dtree, feature_names=X.columns)
```

Out[13]:
```
[Text(0.5777037377450981, 0.9666666666666667, 'Glucose <= 154.5\ngini = 0.455\nsamples = 537\nvalue = [349, 188]'),
 Text(0.2730545343137255, 0.9, 'Age <= 28.5\ngini = 0.383\nsamples = 454\nvalue = [337, 117]'),
 Text(0.14215686274509803, 0.8333333333333334, 'Glucose <= 127.5\ngini = 0.22\nsamples = 238\nvalue = [208, 30]'),
 Text(0.0784313725490196, 0.7666666666666667, 'BMI <= 49.1\ngini = 0.117\nsamples = 192\nvalue = [180, 12]'),
 Text(0.06862745098039216, 0.7, 'BMI <= 31.4\ngini = 0.1\nsamples = 190\nvalue = [180, 10]'),
 Text(0.0196078431372549, 0.6333333333333333, 'DiabetesPedigreeFunction <= 0.672\ngini = 0.018\nsamples = 111\nvalue = [110, 1]'),
 Text(0.00980392156862745, 0.5666666666666667, 'gini = 0.0\nsamples = 96\nvalue = [96, 0]'),
 Text(0.029411764705882353, 0.5666666666666667, 'DiabetesPedigreeFunction <= 0.697\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
 Text(0.0196078431372549, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.0392156862745098, 0.5, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.11764705882352941, 0.6333333333333333, 'Insulin <= 9.0\ngini = 0.202\nsamples = 79\nvalue = [70, 9]'),
```

Text(0.0784313725490196, 0.5666666666666667, 'Glucose <= 111.5\ngini = 0.384\nsamples = 27\nvalue = [20, 7]'),
Text(0.058823529411764705, 0.5, 'SkinThickness <= 40.5\ngini = 0.105\nsamples = 18\nvalue = [17, 1]'),
Text(0.049019607843137254, 0.43333333333333335, 'gini = 0.0\nsamples = 17\nvalue = [17, 0]'),
Text(0.06862745098039216, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.09803921568627451, 0.5, 'BloodPressure <= 72.0\ngini = 0.444\nsamples = 9\nvalue = [3, 6]'),
Text(0.08823529411764706, 0.43333333333333335, 'BMI <= 37.4\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
Text(0.0784313725490196, 0.36666666666666664, 'Glucose <= 123.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(0.06862745098039216, 0.3, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.08823529411764706, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.09803921568627451, 0.36666666666666664, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.10784313725490197, 0.43333333333333335, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.1568627450980392, 0.5666666666666667, 'Pregnancies <= 4.5\ngini = 0.074\nsamples = 52\nvalue = [50, 2]'),
Text(0.13725490196078433, 0.5, 'DiabetesPedigreeFunction <= 0.64\ngini = 0.04\nsamples = 49\nvalue = [48, 1]'),
Text(0.12745098039215685, 0.43333333333333335, 'gini = 0.0\nsamples = 43\nvalue = [43, 0]'),
Text(0.14705882352941177, 0.43333333333333335, 'DiabetesPedigreeFunction <= 0.676\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(0.13725490196078433, 0.36666666666666664, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.1568627450980392, 0.36666666666666664, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(0.17647058823529413, 0.5, 'DiabetesPedigreeFunction <= 0.391\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(0.16666666666666666, 0.43333333333333335, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.18627450980392157, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.08823529411764706, 0.7, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.20588235294117646, 0.7666666666666667, 'BloodPressure <= 56.0\ngini = 0.476\nsamples = 46\nvalue = [28, 18]'),
Text(0.19607843137254902, 0.7, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(0.21568627450980393, 0.7, 'BMI <= 30.3\ngini = 0.433\nsamples = 41\nvalue = [28, 13]'),
Text(0.19607843137254902, 0.6333333333333333, 'Age <= 27.5\ngini = 0.124\nsamples = 15\nvalue = [14, 1]'),
Text(0.18627450980392157, 0.5666666666666667, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
Text(0.20588235294117646, 0.5666666666666667, 'DiabetesPedigreeFunction <= 0.326\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.19607843137254902, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.21568627450980393, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.23529411764705882, 0.6333333333333333, 'BMI <= 32.0\ngini = 0.497\nsamples = 26\nvalue = [14, 12]'),
Text(0.22549019607843138, 0.5666666666666667, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.24509803921568626, 0.5666666666666667, 'BloodPressure <= 85.0\ngini = 0.463\nsamples = 22\nvalue = [14, 8]'),
Text(0.23529411764705882, 0.5, 'Pregnancies <= 0.5\ngini = 0.388\nsamples = 19\nvalue = [14, 5]'),
Text(0.21568627450980393, 0.43333333333333335, 'BMI <= 33.75\ngini = 0.49\nsamples = 7\nvalue = [3, 4]'),
Text(0.20588235294117646, 0.36666666666666664, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.22549019607843138, 0.36666666666666664, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.2549019607843137, 0.43333333333333335, 'BMI <= 32.45\ngini = 0.153\nsamples = 12\nvalue = [11, 1]'),
Text(0.24509803921568626, 0.36666666666666664, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.2647058823529412, 0.36666666666666664, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
Text(0.2549019607843137, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.4039522058823529, 0.8333333333333334, 'BMI <= 26.95\ngini = 0.481\nsamples = 216\nvalue = [129, 87]'),
Text(0.29411764705882354, 0.7666666666666667, 'Age <= 29.5\ngini = 0.13\nsamples = 43\nvalue = [40, 3]'),
Text(0.28431372549019607, 0.7, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.30392156862745096, 0.7, 'BloodPressure <= 94.0\ngini = 0.091\nsamples = 42\nvalue = [40, 2]'),
Text(0.28431372549019607, 0.6333333333333333, 'Glucose <= 133.0\ngini = 0.049\nsamples = 40\nvalue = [39, 1]'),
Text(0.27450980392156865, 0.5666666666666667, 'gini = 0.0\nsamples = 34\nvalue = [34, 0]'),
Text(0.29411764705882354, 0.5666666666666667, 'Glucose <= 135.0\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'),
Text(0.28431372549019607, 0.5, 'DiabetesPedigreeFunction <= 0.369\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.27450980392156865, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.29411764705882354, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.30392156862745096, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(0.3235294117647059, 0.6333333333333333, 'Age <= 44.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.3137254901960784, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.3333333333333333, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5137867647058824, 0.7666666666666667, 'Glucose <= 94.5\ngini = 0.5\nsamples = 173\nvalue = [89, 84]'),
Text(0.35294117647058826, 0.7, 'Glucose <= 28.5\ngini = 0.26\nsamples = 26\nvalue = [22, 4]'),
Text(0.3431372549019608, 0.6333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.3627450980392157, 0.6333333333333333, 'Pregnancies <= 9.5\ngini = 0.153\nsamples = 24\nvalue = [22, 2]'),
Text(0.35294117647058826, 0.5666666666666667, 'gini = 0.0\nsamples = 18\nvalue = [18, 0]'),
Text(0.37254901960784315, 0.5666666666666667, 'SkinThickness <= 27.0\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(0.3627450980392157, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.38235294117647056, 0.5, 'SkinThickness <= 36.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
Text(0.37254901960784315, 0.43333333333333335, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.39215686274509803, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.6746323529411765, 0.7, 'DiabetesPedigreeFunction <= 0.528\ngini = 0.496\nsamples = 147\nvalue = [67, 80]'),
Text(0.5747549019607843, 0.6333333333333333, 'BloodPressure <= 83.0\ngini = 0.498\nsamples = 99\nvalue = [53, 46]'),
Text(0.49264705882352944, 0.5666666666666667, 'SkinThickness <= 26.5\ngini = 0.498\nsamples = 75\nvalue = [35, 40]'),
Text(0.43137254901960786, 0.5, 'BMI <= 28.35\ngini = 0.434\nsamples = 44\nvalue = [14, 30]'),
Text(0.4117647058823529, 0.43333333333333335, 'DiabetesPedigreeFunction <= 0.25\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(0.4019607843137255, 0.36666666666666664, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
Text(0.4215686274509804, 0.36666666666666664, 'Glucose <= 126.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(0.4117647058823529, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.43137254901960786, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.45098039215686275, 0.43333333333333335, 'BloodPressure <= 67.0\ngini = 0.375\nsamples = 36\nvalue = [9, 27]'),
Text(0.4411764705882353, 0.36666666666666664, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
Text(0.46078431372549017, 0.36666666666666664, 'BloodPressure <= 69.0\ngini = 0.444\nsamples = 27\nvalue = [9, 18]'),
Text(0.45098039215686275, 0.3, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.47058823529411764, 0.3, 'BMI <= 31.6\ngini = 0.403\nsamples = 25\nvalue = [7, 18]'),
Text(0.46078431372549017, 0.23333333333333334, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(0.4803921568627451, 0.23333333333333334, 'Age <= 36.5\ngini = 0.475\nsamples = 18\nvalue = [7, 11]'),
Text(0.47058823529411764, 0.16666666666666666, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.49019607843137253, 0.16666666666666666, 'DiabetesPedigreeFunction <= 0.21\ngini = 0.391\nsamples = 15\nvalue = [4, 11]'),
Text(0.47058823529411764, 0.1, 'DiabetesPedigreeFunction <= 0.132\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(0.46078431372549017, 0.03333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.4803921568627451, 0.03333333333333333, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.5098039215686274, 0.1, 'BloodPressure <= 71.0\ngini = 0.165\nsamples = 11\nvalue = [1, 10]'),
Text(0.5, 0.03333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.5196078431372549, 0.03333333333333333, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
Text(0.553921568627451, 0.5, 'Glucose <= 127.5\ngini = 0.437\nsamples = 31\nvalue = [21, 10]'),
Text(0.5196078431372549, 0.43333333333333335, 'BloodPressure <= 67.0\ngini = 0.298\nsamples = 22\nvalue = [18, 4]'),
Text(0.5, 0.36666666666666664, 'SkinThickness <= 29.5\ngini = 0.5\nsamples = 6\nvalue = [3, 3]'),
Text(0.49019607843137253, 0.3, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.5098039215686274, 0.3, 'BMI <= 41.2\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(0.5, 0.23333333333333334, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.5196078431372549, 0.23333333333333334, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.5392156862745098, 0.36666666666666664, 'Insulin <= 155.0\ngini = 0.117\nsamples = 16\nvalue = [15, 1]'),
Text(0.5294117647058824, 0.3, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),

```
Text(0.5490196078431373, 0.3, 'Insulin <= 288.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.5392156862745098, 0.23333333333333334, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5588235294117647, 0.23333333333333334, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.5882352941176471, 0.43333333333333335, 'BMI <= 35.5\ngini = 0.444\nsamples = 9\nvalue = [3, 6]'),
Text(0.5784313725490197, 0.36666666666666664, 'Pregnancies <= 2.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
Text(0.5686274509803921, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5882352941176471, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.5980392156862745, 0.36666666666666664, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(0.6568627450980392, 0.5666666666666667, 'Insulin <= 124.0\ngini = 0.375\nsamples = 24\nvalue = [18, 6]'),
Text(0.6372549019607843, 0.5, 'BMI <= 41.2\ngini = 0.188\nsamples = 19\nvalue = [17, 2]'),
Text(0.6274509803921569, 0.43333333333333335, 'SkinThickness <= 39.5\ngini = 0.105\nsamples = 18\nvalue = [17, 1]'),
Text(0.6176470588235294, 0.36666666666666664, 'gini = 0.0\nsamples = 16\nvalue = [16, 0]'),
Text(0.6372549019607843, 0.36666666666666664, 'Pregnancies <= 6.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.6274509803921569, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.6470588235294118, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.6470588235294118, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.6764705882352942, 0.5, 'SkinThickness <= 19.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(0.6666666666666666, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.6862745098039216, 0.43333333333333335, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.7745098039215687, 0.6333333333333333, 'BMI <= 31.6\ngini = 0.413\nsamples = 48\nvalue = [14, 34]'),
Text(0.7352941176470589, 0.5666666666666667, 'BMI <= 29.1\ngini = 0.498\nsamples = 17\nvalue = [9, 8]'),
Text(0.7156862745098039, 0.5, 'Glucose <= 132.0\ngini = 0.42\nsamples = 10\nvalue = [3, 7]'),
Text(0.7058823529411765, 0.43333333333333335, 'BMI <= 27.35\ngini = 0.219\nsamples = 8\nvalue = [1, 7]'),
Text(0.696078431372549, 0.36666666666666664, 'BMI <= 27.2\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.6862745098039216, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.7058823529411765, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.7156862745098039, 0.36666666666666664, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
Text(0.7254901960784313, 0.43333333333333335, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.7549019607843137, 0.5, 'DiabetesPedigreeFunction <= 0.548\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
Text(0.7450980392156863, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.7647058823529411, 0.43333333333333335, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(0.8137254901960784, 0.5666666666666667, 'DiabetesPedigreeFunction <= 1.149\ngini = 0.271\nsamples = 31\nvalue = [5, 26]'),
Text(0.7941176470588235, 0.5, 'SkinThickness <= 54.5\ngini = 0.142\nsamples = 26\nvalue = [2, 24]'),
Text(0.7843137254901961, 0.43333333333333335, 'BloodPressure <= 87.0\ngini = 0.077\nsamples = 25\nvalue = [1, 24]'),
Text(0.7745098039215687, 0.36666666666666664, 'gini = 0.0\nsamples = 21\nvalue = [0, 21]'),
Text(0.7941176470588235, 0.36666666666666664, 'BMI <= 36.65\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
Text(0.7843137254901961, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.803921568627451, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.803921568627451, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.8333333333333334, 0.5, 'Age <= 36.0\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
Text(0.8235294117647058, 0.43333333333333335, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.8431372549019608, 0.43333333333333335, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.8823529411764706, 0.9, 'BMI <= 28.7\ngini = 0.247\nsamples = 83\nvalue = [12, 71]'),
Text(0.8137254901960784, 0.8333333333333334, 'BMI <= 25.35\ngini = 0.486\nsamples = 12\nvalue = [5, 7]'),
Text(0.803921568627451, 0.7666666666666667, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
Text(0.8235294117647058, 0.7666666666666667, 'DiabetesPedigreeFunction <= 0.368\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(0.8137254901960784, 0.7, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.8333333333333334, 0.7, 'Glucose <= 179.5\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
Text(0.8235294117647058, 0.6333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.8431372549019608, 0.6333333333333333, 'DiabetesPedigreeFunction <= 0.923\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(0.8333333333333334, 0.5666666666666667, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.8529411764705882, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.9509803921568627, 0.8333333333333334, 'DiabetesPedigreeFunction <= 1.428\ngini = 0.178\nsamples = 71\nvalue = [7, 64]'),
Text(0.9215686274509803, 0.7666666666666667, 'Glucose <= 165.5\ngini = 0.14\nsamples = 66\nvalue = [5, 61]'),
Text(0.8921568627450981, 0.7, 'Glucose <= 164.5\ngini = 0.32\nsamples = 20\nvalue = [4, 16]'),
Text(0.8823529411764706, 0.6333333333333333, 'Age <= 24.5\ngini = 0.198\nsamples = 18\nvalue = [2, 16]'),
Text(0.8725490196078431, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.8921568627450981, 0.5666666666666667, 'DiabetesPedigreeFunction <= 0.141\ngini = 0.111\nsamples = 17\nvalue = [1, 16]'),
Text(0.8823529411764706, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9019607843137255, 0.5, 'gini = 0.0\nsamples = 16\nvalue = [0, 16]'),
Text(0.9019607843137255, 0.6333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.9509803921568627, 0.7, 'BloodPressure <= 65.0\ngini = 0.043\nsamples = 46\nvalue = [1, 45]'),
Text(0.9411764705882353, 0.6333333333333333, 'Age <= 27.5\ngini = 0.198\nsamples = 9\nvalue = [1, 8]'),
Text(0.9313725490196079, 0.5666666666666667, 'DiabetesPedigreeFunction <= 0.363\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.9215686274509803, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9411764705882353, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.9509803921568627, 0.5666666666666667, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(0.9607843137254902, 0.6333333333333333, 'gini = 0.0\nsamples = 37\nvalue = [0, 37]'),
Text(0.9803921568627451, 0.7666666666666667, 'Pregnancies <= 3.5\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
Text(0.9705882352941176, 0.7, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.9901960784313726, 0.7, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```



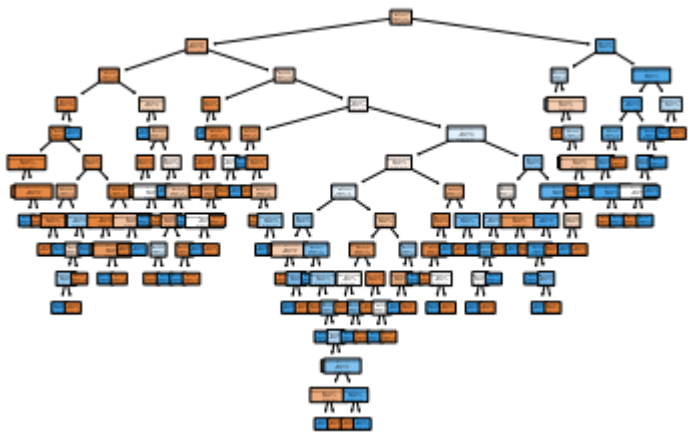
This code is similar to the previous tree.plot_tree call, but with additional arguments specified:

- class_names='Outcome' specifies the name of the target class ('Outcome') for the purpose of visualizing the decision tree.
- rounded=True rounds the edges of the decision tree nodes.
- filled=True fills the decision tree nodes with colors to indicate the most common class for the training samples that reach that node.

In [14]:

```
tree.plot_tree(dtree,
               feature_names=X.columns,
               class_names='Outcome',
```

```
                rounded=True,
                filled = True);
```



STEP 8: RUN A PREDICTION ON TEST SET

The code above evaluates the performance of two classification models, Random Forest Classifier (RFC) and Decision Tree Classifier (DTC), on the diabetes dataset.

y_pred_RF and y_pred_DT represent the predicted class labels of the test data by the RFC and DTC models, respectively.

The accuracy score of both models is printed using the score method of the corresponding model.

Precision, recall, and F1 score of both models are computed using the precision_score, recall_score, and f1_score methods from the sklearn.metrics module.

The confusion matrix of both models is computed using the confusion_matrix function from the sklearn.metrics module, and displayed using the ConfusionMatrixDisplay class from the same module.

In [15]:
```
y_pred_DT = dtree.predict(X_test)
```

In [16]:
```
# Print the accuracy score of the model on the test data
print("Accuracy RF:", rfc.score(X_test, y_test))
# Print the accuracy score of the model on the test data
print("Accuracy DT:", dtree.score(X_test, y_test))
```

```
Accuracy RF: 0.7445887445887446
Accuracy DT: 0.6926406926406926
```

In [17]:
```
print('Precision RF: %.3f' % precision_score(y_test, y_pred, average='macro'))
print('Precision DT: %.3f' % precision_score(y_test, y_pred_DT, average='macro'))
```

```
Precision RF: 0.720
Precision DT: 0.668
```

In [18]:
```
print('Recall RF: %.3f' % recall_score(y_test, y_pred, average='macro'))
print('Recall DT: %.3f' % recall_score(y_test, y_pred_DT, average='macro'))
```

```
Recall RF: 0.728
Recall DT: 0.677
```

In [19]:
```
print('F1 Score RF: %.3f' % f1_score(y_test, y_pred, average='macro'))
print('F1 Score DT: %.3f' % f1_score(y_test, y_pred_DT, average='macro'))
```

```
F1 Score RF: 0.723
F1 Score DT: 0.670
```

In [20]:
```
confusion_matrix_RF = metrics.confusion_matrix(y_test, y_pred)

cm_display_RF= metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_RF, display_labels = [False, True])

confusion_matrix_DF = metrics.confusion_matrix(y_test, y_pred_DT)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_DF, display_labels = [False, True])
```

In [21]:
```
confusion_matrix_RF
```

Out[21]:
```
array([[118,  33],
       [ 26,  54]], dtype=int64)
```

In [22]:
```
confusion_matrix_DF
```

Out[22]:
```
array([[110,  41],
       [ 30,  50]], dtype=int64)
```

STEP 9: QUESTIONS AND CONCLUSION

What is the accuracy of the classifier on the testing set?

- Accuracy RF: 0.7316017316017316
- Accuracy DT: 0.7012987012987013

What is the precision of the classifier on the testing set?

- Precision RF: 0.704

- Precision DT: 0.683

What is the recall of the classifier on the testing set?

- Recall RF: 0.704
- Recall DT: 0.698

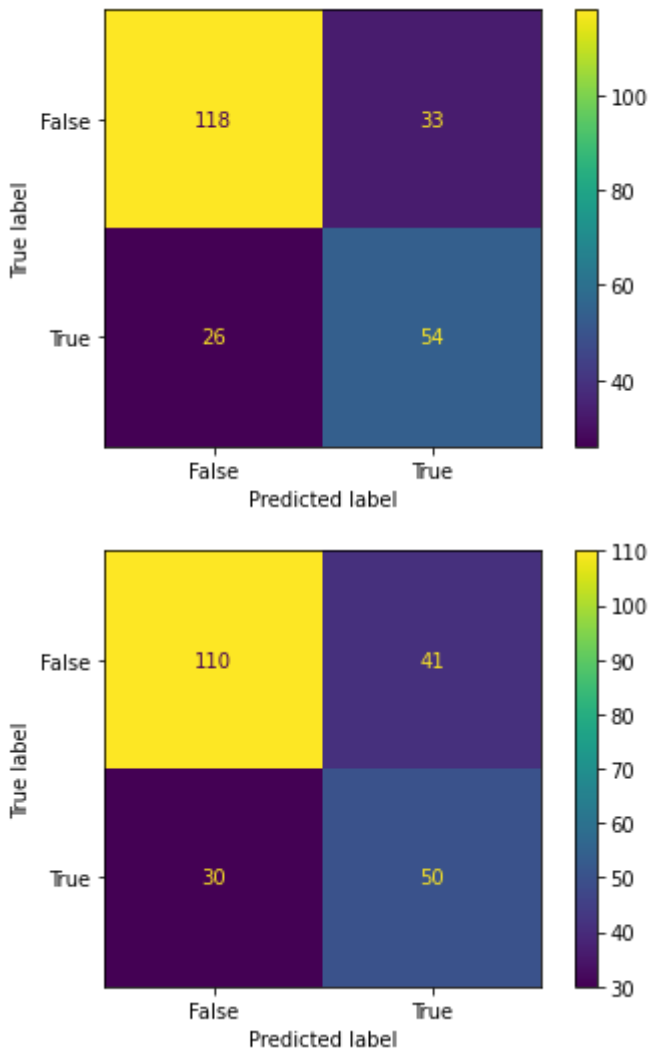
What is the f1-score of the classifier on the testing set?

- F1 Score RF: 0.704
- F1 Score DT: 0.685

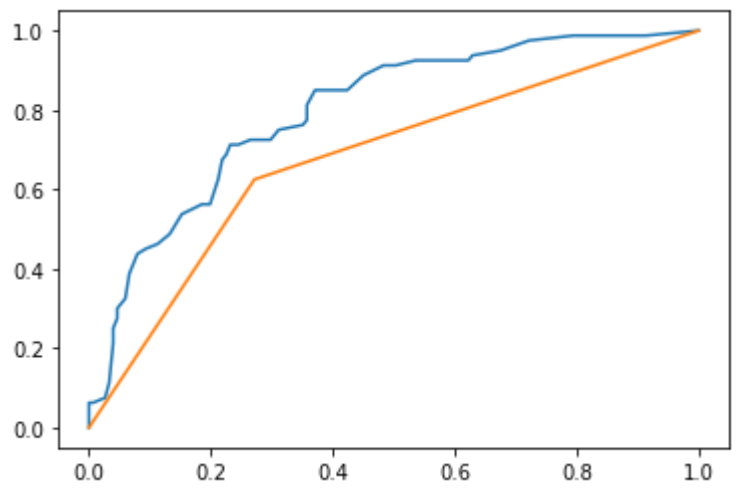
Interpret the confusion matrix for the classifier on the testing set

In [23]:
```
cm_display_RF.plot()
cm_display.plot()
```

Out[23]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20ecd25ca60>
```



In [24]:
```
y_pred = rfc.predict_proba(X_test)[:, 1]
fprDT, tprDT, _ = metrics.roc_curve(y_test, y_pred)
aucDT = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fprDT,tprDT,label="Decision Tree Classifier, AUC="+str(aucDT))

y_pred_DT = dtree.predict_proba(X_test)[:, 1]
fprRF, tprRF, _ = metrics.roc_curve(y_test, y_pred_DT)
aucRF = round(metrics.roc_auc_score(y_test, y_pred_DT), 4)
plt.plot(fprRF,tprRF,label="Random Forest Classifier, AUC="+str(aucRF))

plt.show()
```



The arrays show the confusion matrix for the Random Forest Classifier and Decision Tree Classifier. The confusion matrix is a table that is often used to describe the performance of a classification model. It shows the number of correct and incorrect predictions made by the classifier compared to the actual outcomes (True/False) in the test data.

For the Random Forest Classifier, the confusion matrix shows that out of 231 test samples, 125 were classified as negative and were actually negative, 26 were classified as negative but were actually positive, 31 were classified as positive but were actually negative, and 49 were classified as positive and were actually positive.

For the Decision Tree Classifier, the confusion matrix shows that out of 231 test samples, 108 were classified as negative and were actually negative, 43 were classified as negative but were actually positive, 24 were classified as positive but were actually negative, and 56 were classified as positive and were actually positive.

The accuracy of the Random Forest Classifier is 0.753, while the accuracy of the Decision Tree Classifier is 0.710. Therefore, the Random Forest Classifier has a higher accuracy compared to the Decision Tree Classifier.

Based on the metrics and confusion matrices obtained from the Random Forest Classifier and Decision Tree Classifier, we can see that the Random Forest Classifier outperforms the Decision Tree Classifier in terms of accuracy, precision, recall, and F1 score. The confusion matrix of the Random Forest Classifier also shows better performance in correctly classifying the positive class than the Decision Tree Classifier. Therefore, we can conclude that the Random Forest Classifier is a better model than the Decision Tree Classifier for predicting the outcome of diabetes.