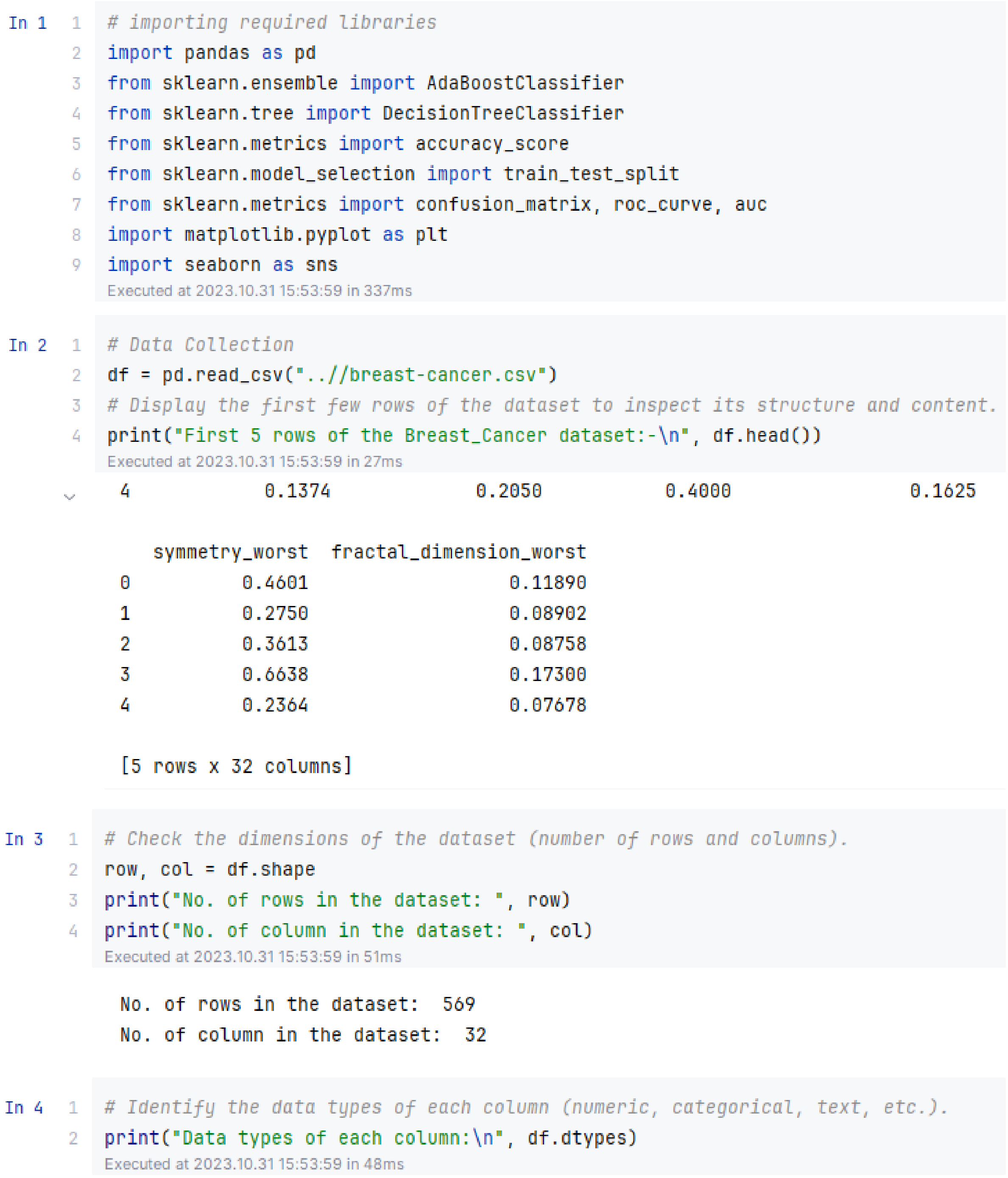
AIM:- Ensemble Learning: Write a program to implement the Adaboost algorithm with decision tree as the base classifier. The decision tree implemented as a function. Run Adaboost for 3 rounds. The combined classifier should be tested on test instances and the accuracy of prediction for the test instances should be printed as output. A single program should train the classifier on the training set as well as test it on the test set.



In 1

```
# importing required libraries
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
```

Executed at 2023.10.31 15:53:59 in 337ms

In 2

```
# Data Collection
df = pd.read_csv("..//breast-cancer.csv")
# Display the first few rows of the dataset to inspect its structure and content.
print("First 5 rows of the Breast_Cancer dataset:-\n", df.head())
```

Executed at 2023.10.31 15:53:59 in 27ms

```
4           0.1374           0.2050           0.4000             0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

In 3

```
# Check the dimensions of the dataset (number of rows and columns).
row, col = df.shape
print("No. of rows in the dataset: ", row)
print("No. of column in the dataset: ", col)
```

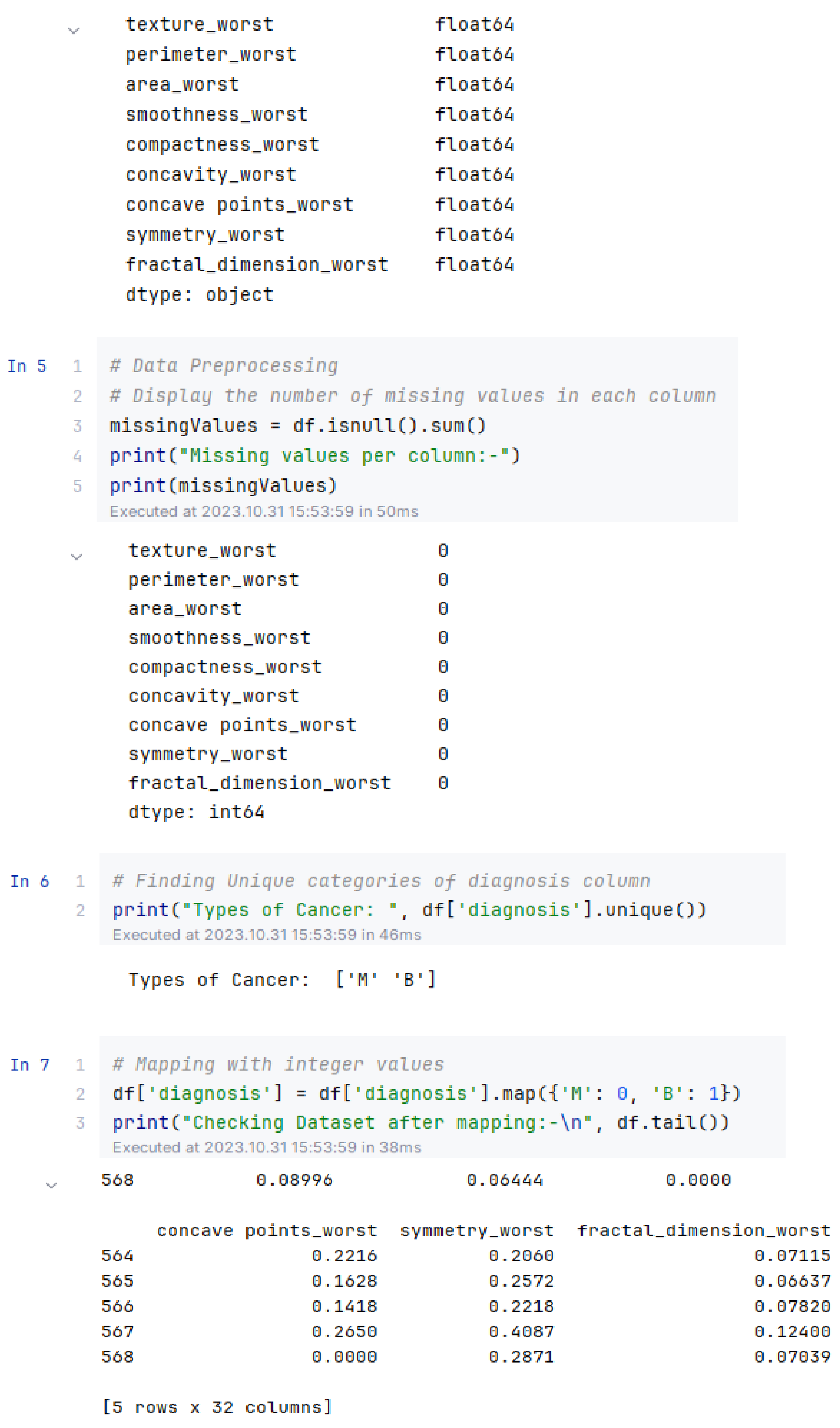
Executed at 2023.10.31 15:53:59 in 51ms

```
No. of rows in the dataset:  569
No. of column in the dataset:  32
```

In 4

```
# Identify the data types of each column (numeric, categorical, text, etc.).
print("Data types of each column:\n", df.dtypes)
```

Executed at 2023.10.31 15:53:59 in 48ms



```
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

In 5

```python
# Data Preprocessing
# Display the number of missing values in each column
missingValues = df.isnull().sum()
print("Missing values per column:-")
print(missingValues)
```

Executed at 2023.10.31 15:53:59 in 50ms

```
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In 6

```python
# Finding Unique categories of diagnosis column
print("Types of Cancer: ", df['diagnosis'].unique())
```

Executed at 2023.10.31 15:53:59 in 46ms

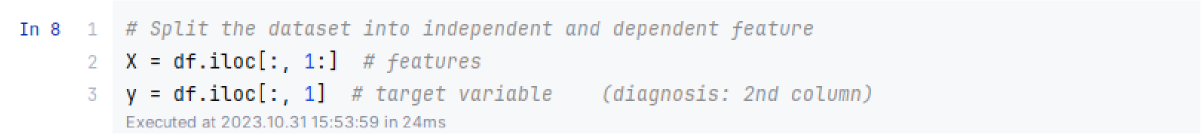
```
Types of Cancer:  ['M' 'B']
```

In 7

```python
# Mapping with integer values
df['diagnosis'] = df['diagnosis'].map({'M': 0, 'B': 1})
print("Checking Dataset after mapping:-\n", df.tail())
```

Executed at 2023.10.31 15:53:59 in 38ms

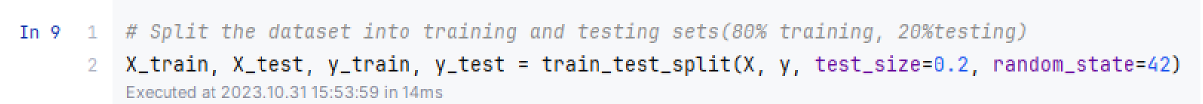
```
568           0.08996            0.06444           0.0000

     concave points_worst  symmetry_worst  fractal_dimension_worst
564                0.2216          0.2060                  0.07115
565                0.1628          0.2572                  0.06637
566                0.1418          0.2218                  0.07820
567                0.2650          0.4087                  0.12400
568                0.0000          0.2871                  0.07039

[5 rows x 32 columns]
```



In 8

```python
# Split the dataset into independent and dependent feature
X = df.iloc[:, 1:]  # features
y = df.iloc[:, 1]  # target variable    (diagnosis: 2nd column)
```

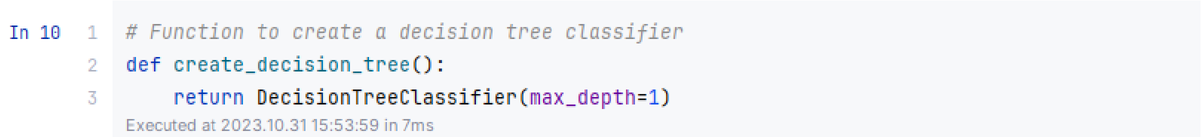
Executed at 2023.10.31 15:53:59 in 24ms



In 9

```python
# Split the dataset into training and testing sets(80% training, 20%testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

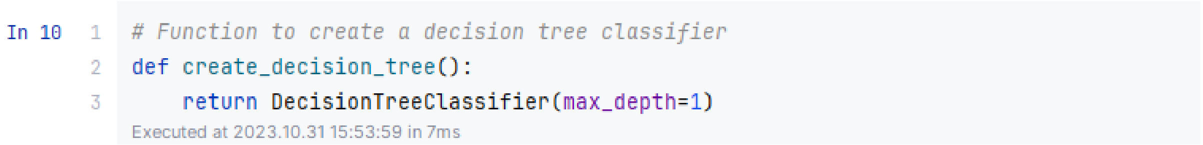
Executed at 2023.10.31 15:53:59 in 14ms



In 10

```python
# Function to create a decision tree classifier
def create_decision_tree():
    return DecisionTreeClassifier(max_depth=1)
```

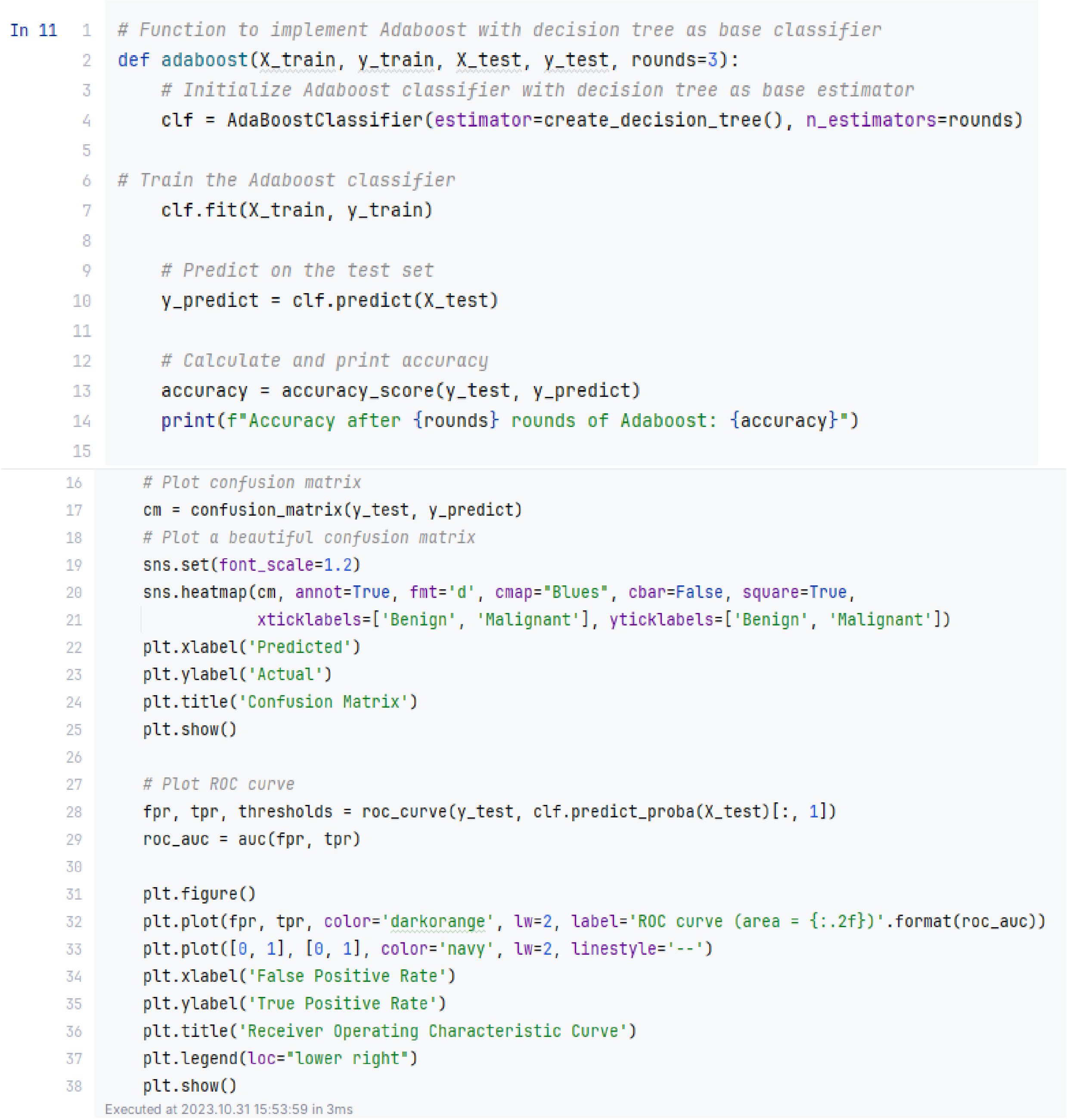
Executed at 2023.10.31 15:53:59 in 7ms



In 10

```python
# Function to create a decision tree classifier
def create_decision_tree():
    return DecisionTreeClassifier(max_depth=1)
```

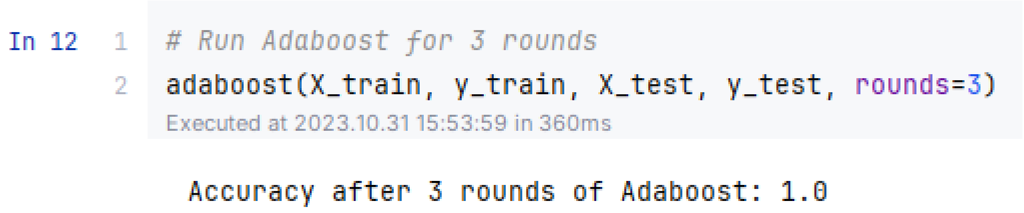
Executed at 2023.10.31 15:53:59 in 7ms



In 11

```python
# Function to implement Adaboost with decision tree as base classifier
def adaboost(X_train, y_train, X_test, y_test, rounds=3):
    # Initialize Adaboost classifier with decision tree as base estimator
    clf = AdaBoostClassifier(estimator=create_decision_tree(), n_estimators=rounds)

# Train the Adaboost classifier
    clf.fit(X_train, y_train)

    # Predict on the test set
    y_predict = clf.predict(X_test)

    # Calculate and print accuracy
    accuracy = accuracy_score(y_test, y_predict)
    print(f"Accuracy after {rounds} rounds of Adaboost: {accuracy}")

    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_predict)
    # Plot a beautiful confusion matrix
    sns.set(font_scale=1.2)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False, square=True,
                xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    # Plot ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend(loc="lower right")
    plt.show()
```

Executed at 2023.10.31 15:53:59 in 3ms

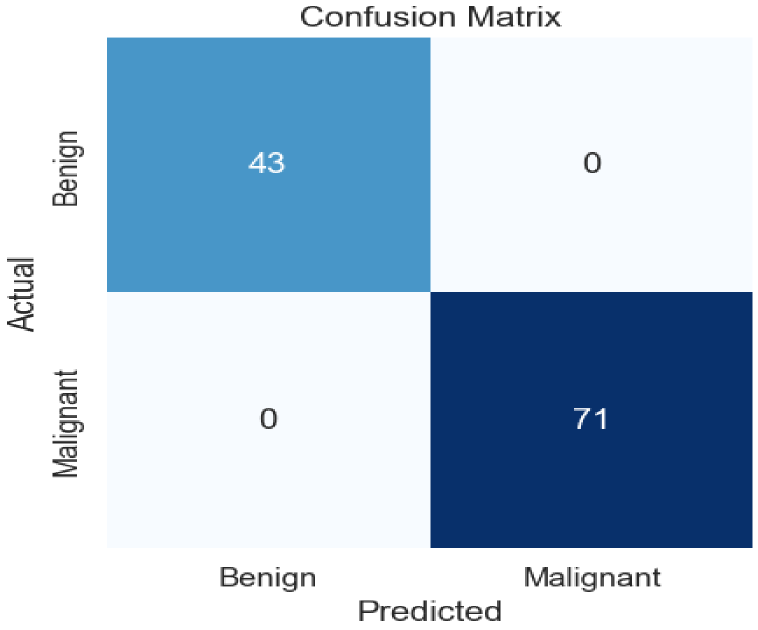


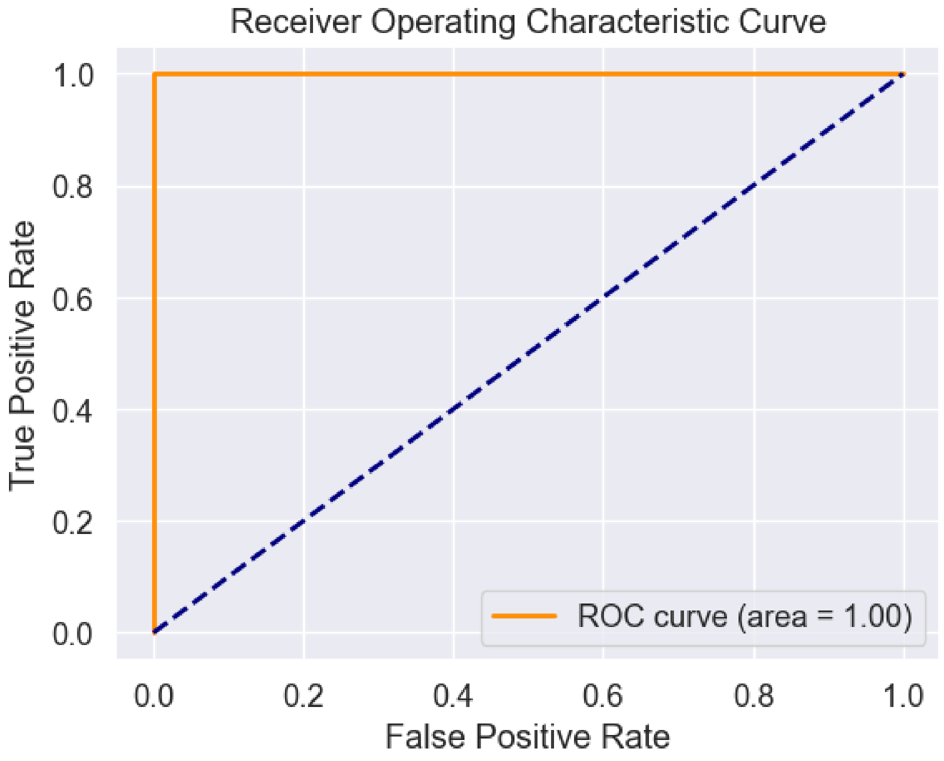
In 12

```
# Run Adaboost for 3 rounds
adaboost(X_train, y_train, X_test, y_test, rounds=3)
```

Executed at 2023.10.31 15:53:59 in 360ms

```
Accuracy after 3 rounds of Adaboost: 1.0
```





Submitted By,
Name:- Shankar Singh Mahanty
Regd. No:- 2101020758
Roll No:- CSE21238
Group:- 3,   Sem:- 5th
Branch:- CSE