BREAST CANCER DIAGNOSIS AND PREDICTION USING THE WISCONSIN BREAST CANCER DIAGNOSTIC DATASET

Library Imports

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
Rf = RandomForestClassifier()
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=12)
from sklearn.svm import SVC
svm = SVC(kernel='linear', random_state = 10)
```

Dataset Upload

In [4]:

```
df = pd.read_csv('C:/Users/USER/Desktop/breast cancer wiscosin.csv')
```

In [5]:

```
# Display of the first 5 rows in the dataset
df.head(5)
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 32 columns

In [6]:

```
#Overview of the Information in the Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

Data Preprocessing

In [7]:

```
#Checking For Missing Values
df.isnull().sum().sort_values(ascending = False)
```

Out[7]:

```
id                         0
diagnosis                  0
symmetry_worst             0
concave points_worst       0
concavity_worst            0
compactness_worst          0
smoothness_worst           0
area_worst                 0
perimeter_worst            0
texture_worst              0
radius_worst               0
fractal_dimension_se       0
symmetry_se                0
concave points_se          0
concavity_se               0
compactness_se             0
smoothness_se              0
area_se                    0
perimeter_se               0
texture_se                 0
radius_se                  0
fractal_dimension_mean     0
symmetry_mean              0
concave points_mean        0
concavity_mean             0
compactness_mean           0
smoothness_mean            0
area_mean                  0
perimeter_mean             0
texture_mean               0
radius_mean                0
fractal_dimension_worst    0
dtype: int64
```

In [8]:

```
#Check for Duplicates
df = df.drop_duplicates()
df.shape
#No duplicates in the Dataset
```

Out[8]:

```
(569, 32)
```

In [9]:

```
#Dropping the ID Column
df =df.drop(['id'], axis =1)
```

In [12]:

```
df.diagnosis.replace({'B':'Benign','M':'Malignant'}, inplace = True)
```

In [13]:

```
df.diagnosis.value_counts()
```

Out[13]:

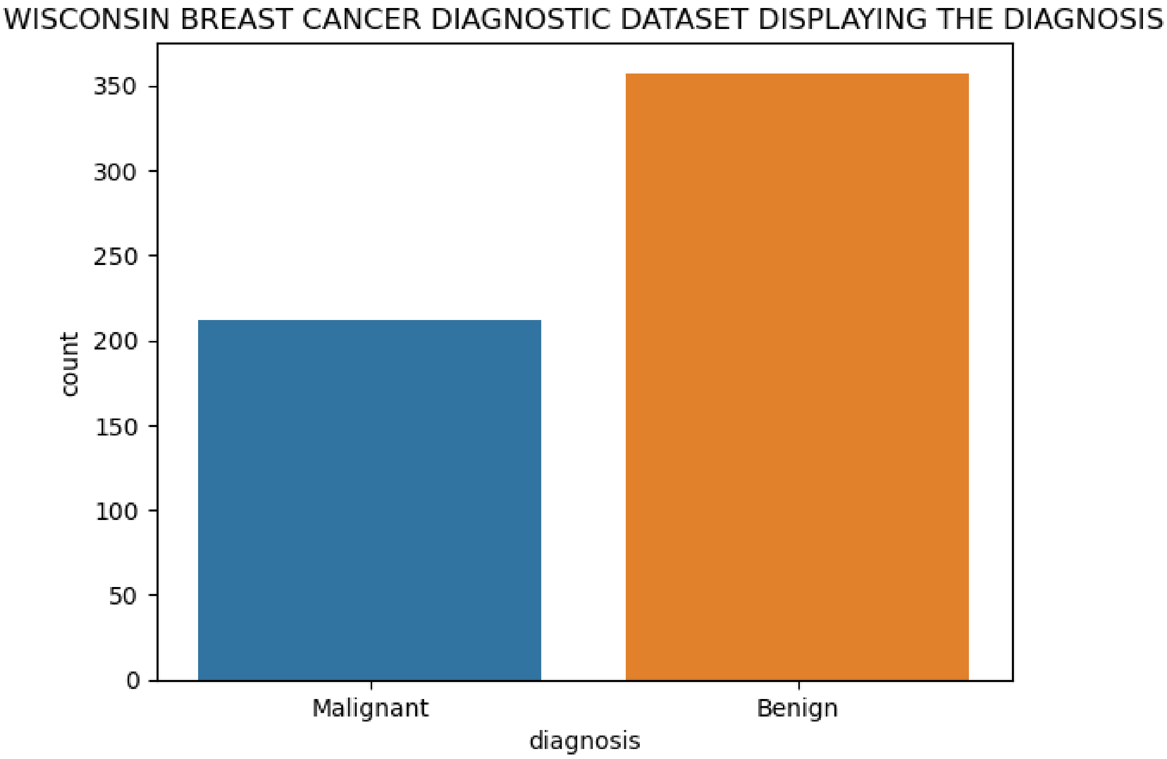
```
Benign       357
Malignant    212
Name: diagnosis, dtype: int64
```

In [14]:

```
sns.countplot(x='diagnosis', data = df)
plt.title('WISCONSIN BREAST CANCER DIAGNOSTIC DATASET DISPLAYING THE DIAGNOSIS')
```

Out[14]:

```
Text(0.5, 1.0, 'WISCONSIN BREAST CANCER DIAGNOSTIC DATASET DISPLAYING THE DIAGNOSI
S')
```



In [28]:

```
#Converting Categorical Data to Ordinal
df.diagnosis.replace({'Benign':0,'Malignant':1}, inplace = True)
```

FEATURE SELECTION WITH CORRELATION MATRIX

In [16]:

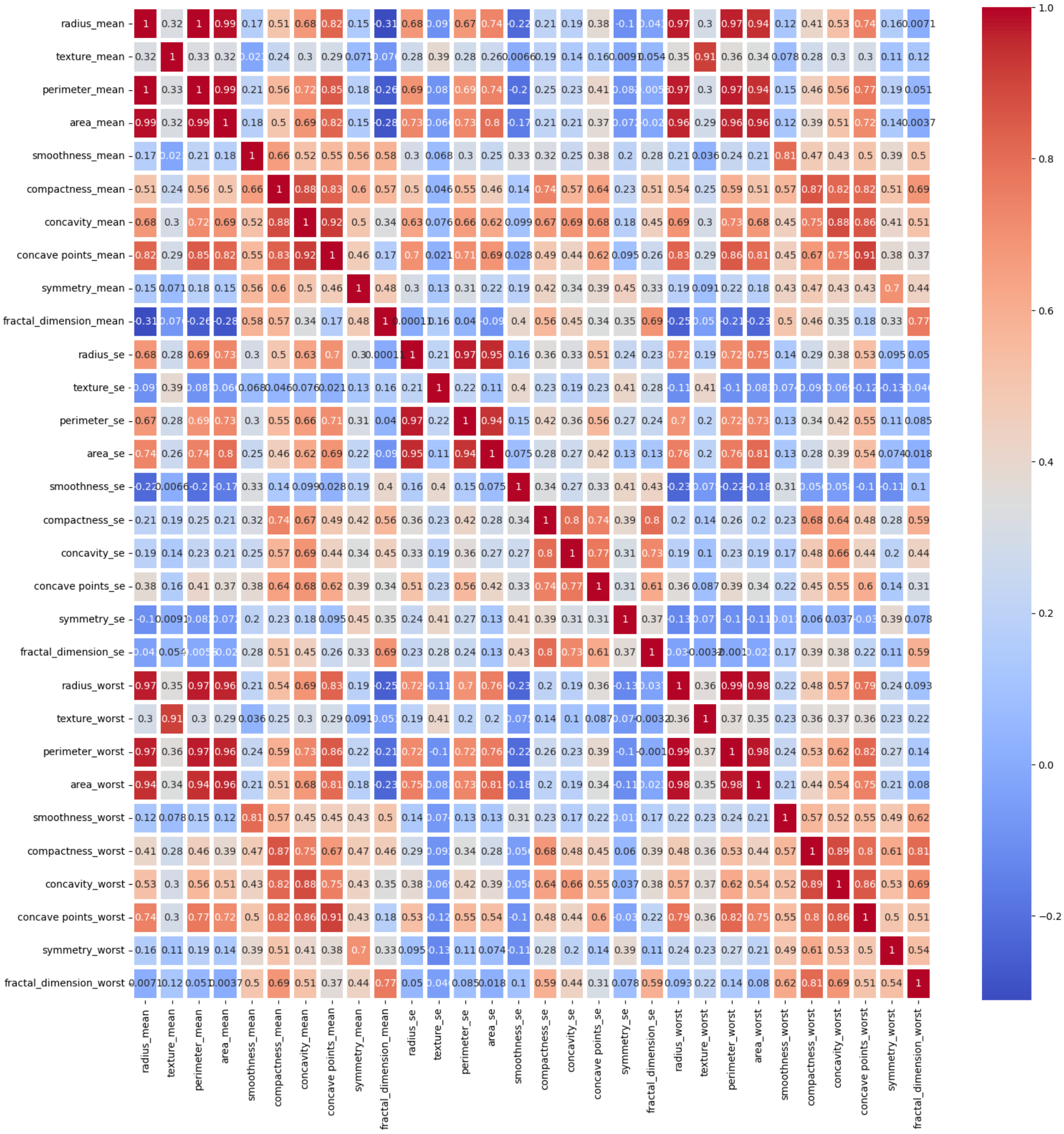
```
correlation_matrix = df.corr()
```

In [17]:

```
plt.figure(figsize=(18, 18))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=3)
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:
```
threshold = 0.8
correlated_pairs = {}

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            correlated_pairs[colname] = correlation_matrix.columns[j]
```

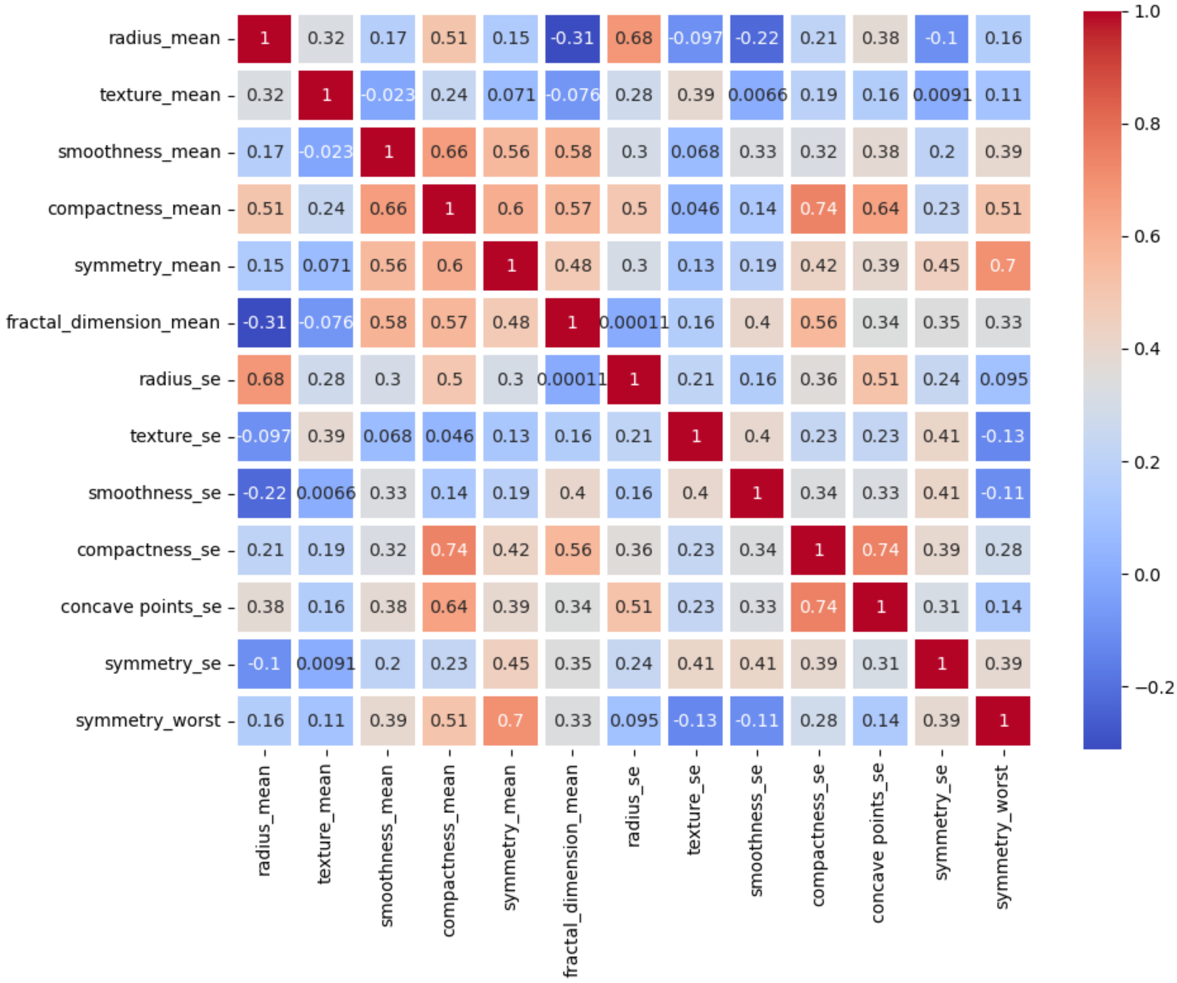
In [19]:
```
#Removal of Select Features With High Correlation
features_to_remove = set(correlated_pairs.keys())
df_reduced = df.drop(columns=features_to_remove)
```

In [20]:
```
#Dataset with reduced features
df_reduced.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 0 to 568
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   diagnosis               569 non-null    object 
 1   radius_mean             569 non-null    float64
 2   texture_mean            569 non-null    float64
 3   smoothness_mean         569 non-null    float64
 4   compactness_mean        569 non-null    float64
 5   symmetry_mean           569 non-null    float64
 6   fractal_dimension_mean  569 non-null    float64
 7   radius_se               569 non-null    float64
 8   texture_se              569 non-null    float64
 9   smoothness_se           569 non-null    float64
 10  compactness_se          569 non-null    float64
 11  concave points_se       569 non-null    float64
 12  symmetry_se             569 non-null    float64
 13  symmetry_worst          569 non-null    float64
dtypes: float64(13), object(1)
memory usage: 66.7+ KB
```

In [21]:

```
#Heatmap displaying the current features and their correlation
cm = df_reduced.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='coolwarm', linewidths=3)
plt.tight_layout()
```



TRAIN - TEST SPLIT (75:25)

In [22]:

```
df_reduced =df_reduced.drop(['diagnosis'], axis =1)
```

In [29]:

```
X = df_reduced
y =df['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

Model Implementation

In [30]:

```
# Defining a function to plot the Receiver Operating Characteristic (ROC) curve
def plot_roc_curve(y_true, y_scores):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    auc = roc_auc_score(y_true, y_scores)
    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc))
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate (FPR)')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()
```

Logistic Regression Model

In [31]:

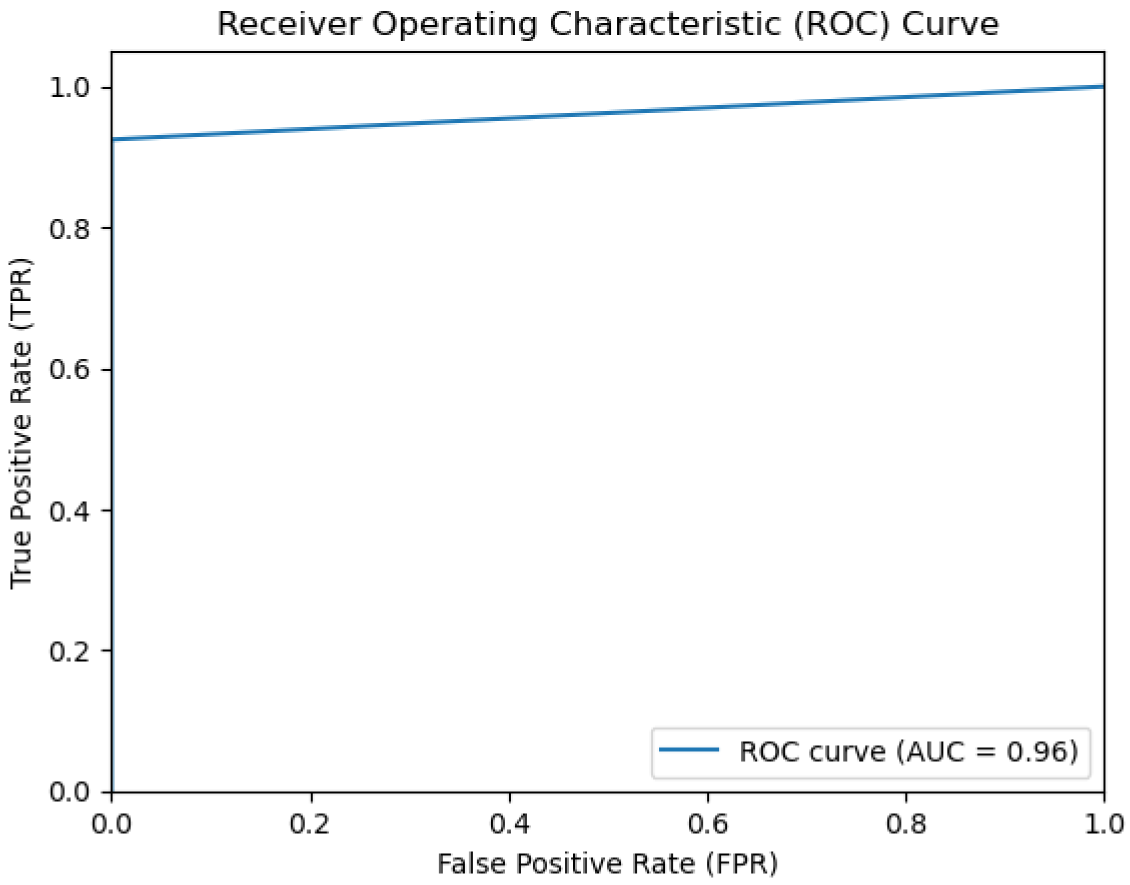
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
predictions = lr.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,predictions)))
plot_roc_curve(y_test, predictions)
```

```
[[90  0]
 [ 4 49]]
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        90
           1       1.00      0.92      0.96        53

    accuracy                           0.97       143
   macro avg       0.98      0.96      0.97       143
weighted avg       0.97      0.97      0.97       143

ROC_AUC Score :  96.23%
```



In [32]:

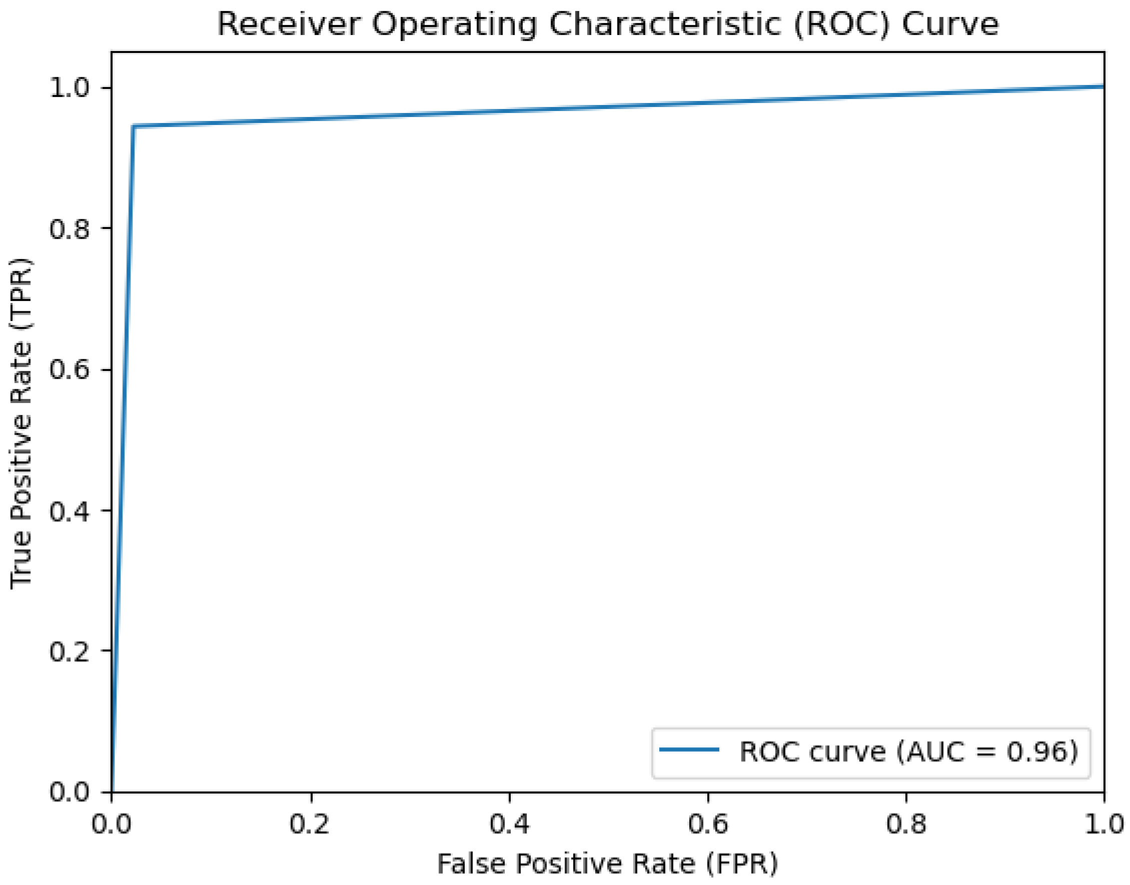
```
Rf.fit(X_train, y_train)
predictions = Rf.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,predictions)))
plot_roc_curve(y_test, predictions)
```

```
[[88  2]
 [ 3 50]]
              precision    recall  f1-score   support

           0       0.97      0.98      0.97        90
           1       0.96      0.94      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.96      0.97      0.96       143

ROC_AUC Score :  96.06%
```



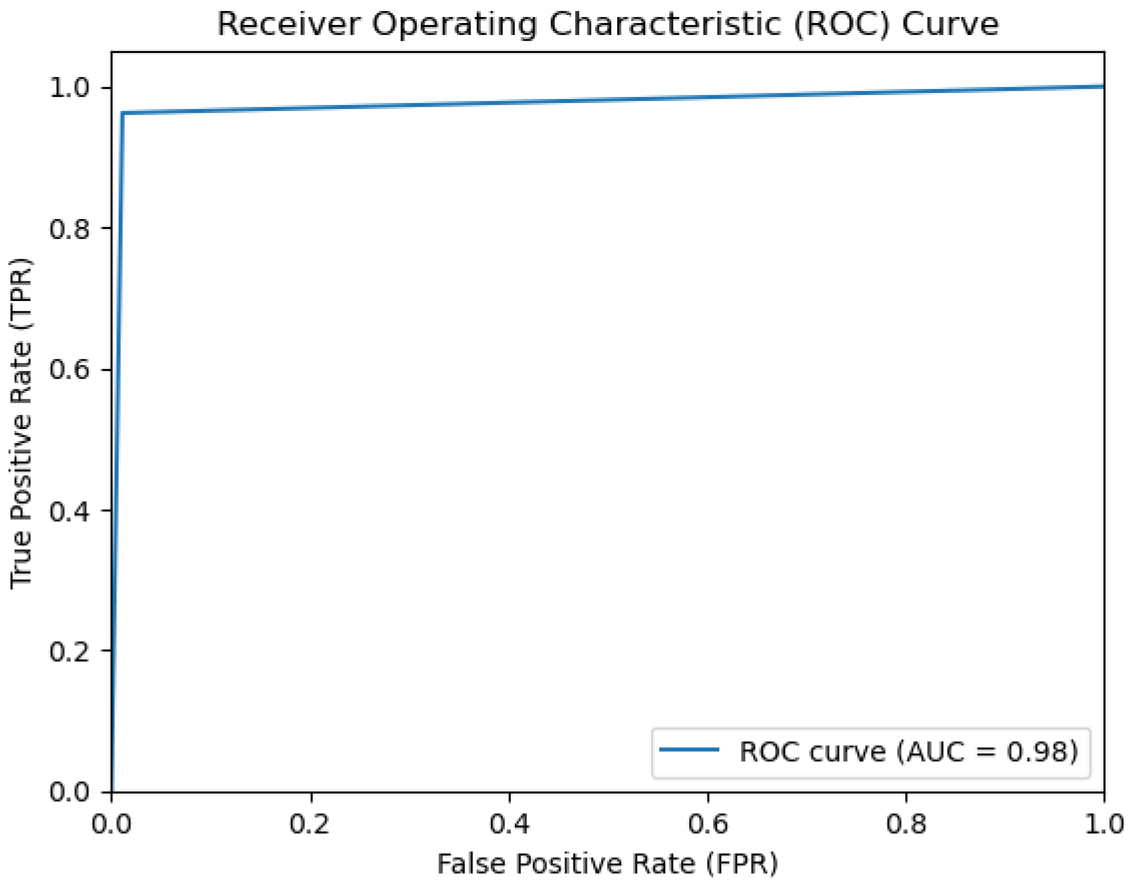
In [33]:
```
svm.fit(X_train, y_train)
predictions = svm.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,predictions)))
plot_roc_curve(y_test, predictions)
```

```
[[89  1]
 [ 2 51]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.98        90
           1       0.98      0.96      0.97        53

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143

ROC_AUC Score :  97.56%
```



In [34]:

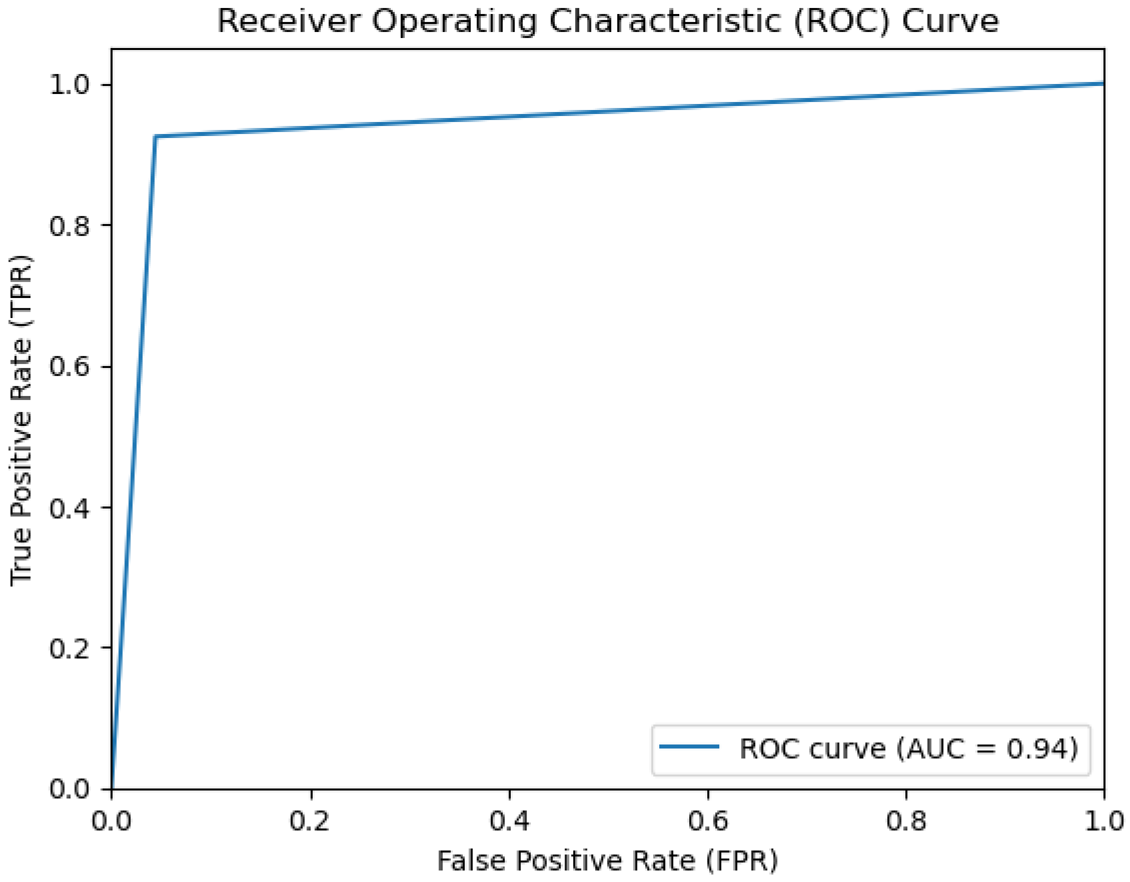
```
Dt_model = DecisionTreeClassifier(max_depth = 7)
Dt_model.fit(X_train, y_train)
predictions = Dt_model.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,predictions)))
plot_roc_curve(y_test, predictions)
```

```
[[86  4]
 [ 4 49]]
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        90
           1       0.92      0.92      0.92        53

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143

ROC_AUC Score :  94.00%
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.94)