Gradient boosting

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [4]:

```
dataset = pd.read_csv('/Users/myyntiimac/Desktop/Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

In [5]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0
```

In [7]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [9]:

```
classifier=GradientBoostingClassifier()
```

In [10]:

```
classifier.fit(X_train, y_train)
```

Out[10]:

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [11]:

```
y_pred = classifier.predict(X_test)
```

In [12]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[64  4]
 [ 5 27]]
```

In [14]:

```
from sklearn.metrics import accuracy_score
# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[14]:

```
0.91
```

In [15]:

```
bias = classifier.score(X_train, y_train)
bias
```

Out[15]:

```
0.9733333333333334
```

In [16]:

```
variance = classifier.score(X_test, y_test)
variance
```

Out[16]:

```
0.91
```

In [17]:

```
#ROC AND AUC
from sklearn.metrics import roc_curve, roc_auc_score
```

In [18]:

```
# Compute the False Positive Rate (FPR), True Positive Rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [19]:

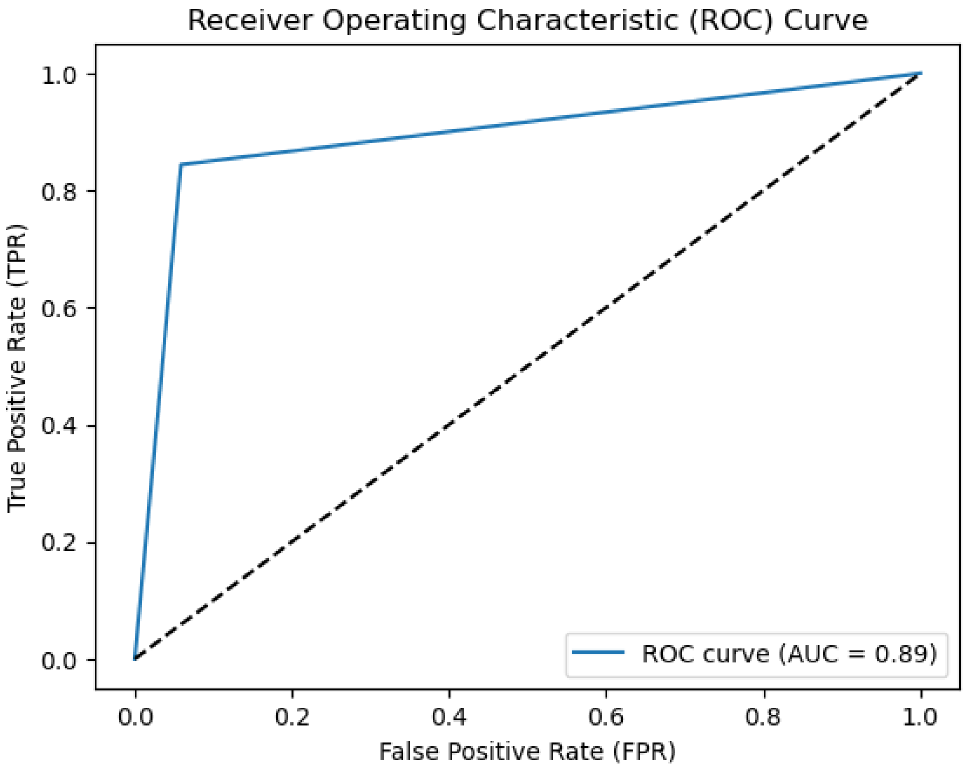
```
auc = roc_auc_score(y_test, y_pred)
auc
```

Out[19]:

```
0.8924632352941176
```

In [20]:

```
# Plotting the ROC curve
plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], 'k--')  # Random guess line
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()
```




```
Insight:In the case of AUC = 0.89, the model demonstrates reasonable discriminative
ability, but there is still room for improvement. It correctly ranks 89% of the positive
samples higher than the negative samples, on average, across different classification
thresholds. However, it might misclassify some instances, leading to false positives or
false negatives.
```

In [21]:

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 88.33 %
Standard Deviation: 5.43 %
```

In [22]:

```
from sklearn.model_selection import GridSearchCV
```

In [30]:

```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees
    'learning_rate': [0.1, 0.01, 0.001],  # Learning rate
    'max_depth': [3, 5, 7],  # Maximum depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required at a leaf node
    'subsample': [0.8, 1.0],  # Fraction of samples to be used for fitting each tree
    'max_features': [1.0, 'sqrt']  # Number of features to consider for the best split
}
```

In [32]:

```
# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=classifier, param_grid=param_grid, cv=10)
grid_search = GridSearchCV(classifier, param_grid, cv=10)
grid_search = grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)

```

```
Best Accuracy: 90.67 %
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'max_features': 'sqrt', 'm
in_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 0.8}
```

In [ ]:

```

```