Q1) Use the `train_test_split` function to split the `features` and `Y` dataframes with a `test_size` of `0.2` and the `random_state` set to `10`.

```
#Enter Your Code and Execute
X = preprocessing.StandardScaler().fit(features).transform(features)
Y = Y.astype(int)
```

[11] ✓ 0.0s Python

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=10)
```

[12] ✓ 0.0s Python

Q2) Create and train a Linear Regression model called LinearReg using the training data (`x_train`, `y_train`).

```
#Enter Your Code and Execute
LinearReg = LinearRegression()
LinearReg
```

[13] ✓ 0.0s Python

```
LinearRegression
LinearRegression()
```

```
LinearReg = LinearReg.fit(x_train, y_train)
```

[14] ✓ 0.3s Python

Q3) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions`.

```
#Enter Your Code and Execute
predictions = LinearReg.predict(x_test)
```

[15] ✓ 0.0s Python

```
predictions
```

[16] ✓ 0.0s Python

```
array([ 1.31202841e-01,  2.75245809e-01,  9.77333212e-01,  2.87086630e-01,
        1.31141806e-01,  4.60365438e-01,  3.58131552e-01,  8.57521200e-01,
        6.73866415e-01,  3.75749110e-02,  4.18868050e-03,  2.81959677e-01,
        3.38173056e-01,  7.92008875e-02,  6.24162196e-02,  5.64247274e-01,
       -6.26448156e-02,  5.22071981e-01,  1.51405477e-01,  3.58863973e-01,
        6.02799891e-02,  9.02809286e-01,  4.67933798e-01,  2.02186727e-01,
       -7.22883703e-02,  3.84620809e-01,  5.35316610e-01, -2.31550695e-02,
        6.39473104e-01, -9.62141515e-02,  3.78090048e-01,  1.17958212e-01,
```

Q4) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

+ Code + Markdown

```
#Enter Your Code and Execute
LinearRegression_MAE = metrics.mean_absolute_error(y_test, predictions)
LinearRegression_MSE = metrics.mean_squared_error(y_test, predictions)
LinearRegression_R2 = metrics.r2_score(y_test, predictions)
```

[17] ✓ 0.0s Python

```
print("Mean Absolute Error: ", LinearRegression_MAE)
print("Mean Squared Error: ", LinearRegression_MSE)
print("R2 Score: ", LinearRegression_R2)
```

[18] ✓ 0.0s Python

```
Mean Absolute Error:  0.2561684217367737
Mean Squared Error:  0.11567081994779356
R2 Score:  0.42737845555129983
```

Q5) Show the MAE, MSE, and R2 in a tabular format using data frame for the linear model.

```
#Enter Your Code and Execute
Report = pd.DataFrame(data=[['Linear Regression', LinearRegression_MAE, LinearRegression_MSE, LinearRegression_R2]],
                      columns=['Model', 'Mean Absolute Error', 'Mean Squared Error', 'R2 Score'])
```

[19] ✓ 0.0s Python

```
Report
```

[20] ✓ 0.0s Python

	Model	Mean Absolute Error	Mean Squared Error	R2 Score
0	Linear Regression	0.256168	0.115671	0.427378

Q6) Create and train a KNN model called KNN using the training data (`x_train`, `y_train`) with the `n_neighbors` parameter set to `4`.

```
#Enter Your Code and Execute
KNN = KNeighborsClassifier(n_neighbors=4)
KNN
```

[21] Python

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

```
KNN = KNN.fit(x_train, y_train)
```

[22] Python

Q7) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions`.

```
[23]
#Enter Your Code and Execute
predictions = KNN.predict(x_test)
```

✓ 0.3s Python

```
[24]
predictions
```

✓ 0.0s Python

```
array([0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 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, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Q8) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

```python
#Enter Your Code and Execute
KNN_Accuracy_Score = accuracy_score(y_test, predictions)
KNN_JaccardIndex = jaccard_score(y_test, predictions)
KNN_F1_Score = f1_score(y_test, predictions)
```

[25] ✓ 0.0s Python

```python
print("Accuracy Score: ", KNN_Accuracy_Score)
print("Jaccard Index: ", KNN_JaccardIndex)
print("F1 Score: ", KNN_F1_Score)
```

[26] ✓ 0.0s Python

```
Accuracy Score:  0.7603053435114504
Jaccard Index:  0.24154589371980675
F1 Score:  0.38910505836575876
```

Q9) Create and train a Decision Tree model called Tree using the training data (`x_train`, `y_train`).

[27]

```
#Enter Your Code and Execute
Tree = DecisionTreeClassifier(criterion="entropy", max_depth=6)
Tree
```

Python

```
DecisionTreeClassifier(criterion='entropy', max_depth=6)
```

[28]

```
Tree = Tree.fit(x_train, y_train)
```

Python

Q10) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions`.

```
#Enter Your Code and Execute
predictions = Tree.predict(x_test)
```

[29] ✓ 0.0s Python

+ Code + Markdown

```
predictions
```

[30] ✓ 0.0s Python

```
array([0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
```

Q11) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

+ Code + Markdown

```
#Enter Your Code and Execute
Tree_Accuracy_Score = accuracy_score(y_test, predictions)
Tree_JaccardIndex = jaccard_score(y_test, predictions)
Tree_F1_Score = f1_score(y_test, predictions)
```

[31] ✓ 0.0s Python

```
print("Accuracy Score: ", Tree_Accuracy_Score)
print("Jaccard Index: ", Tree_JaccardIndex)
print("F1 Score: ", Tree_F1_Score)
```

[32] ✓ 0.0s Python

```
Accuracy Score:  0.8259541984732824
Jaccard Index:  0.49557522123893805
F1 Score:  0.6627218934911243
```

Q12) Use the `train_test_split` function to split the `features` and `Y` dataframes with a `test_size` of `0.2` and the `random_state` set to `10`.

```python
#Enter Your Code and Execute
X = preprocessing.StandardScaler().fit(features).transform(features)
Y = Y.astype(int)
```

[33] Python

```python
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=10)
```

[34] Python

Q13) Create and train a LogisticRegression model called LR using the training data (`x_train`, `y_train`) with the `solver` parameter set to `liblinear`.

```
#Enter Your Code and Execute
LR = LogisticRegression(C=0.01, solver='liblinear')
LR
```

[35] ✓ 0.0s Python

```
LogisticRegression
LogisticRegression(C=0.01, solver='liblinear')
```

```
LR = LR.fit(x_train, y_train)
```

[36] ✓ 0.0s Python

Q14) Now, use the `predict` and `predict_proba` methods on the testing data (`x_test`) and save it as 2 arrays `predictions` and `predict_proba`.

```
#Enter Your Code and Execute
predictions = LR.predict(x_test)
```

[37] ✓ 0.0s Python

```
predictions
```

[ ] Python

```
predict_proba = LR.predict_proba(x_test)
```

[39] ✓ 0.0s Python

Q15) Using the `predictions`, `predict_proba` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

```
#Enter Your Code and Execute
LR_Accuracy_Score = accuracy_score(y_test, predictions)
LR_JaccardIndex = jaccard_score(y_test, predictions)
LR_F1_Score = f1_score(y_test, predictions)
LR_Log_Loss = log_loss(y_test, predict_proba)
```

[40] ✓ 0.0s Python

```
print("Accuracy Score: ", LR_Accuracy_Score)
print("Jaccard Index: ", LR_JaccardIndex)
print("F1 Score: ", LR_F1_Score)
print("Log Loss: ", LR_Log_Loss)
```

[41] ✓ 0.0s Python

```
Accuracy Score:  0.8442748091603054
Jaccard Index:  0.5426008968609866
F1 Score:  0.7034883720930233
Log Loss:  0.37287610286439166
```

Q16) Create and train a SVM model called SVM using the training data (`x_train`, `y_train`).

```
#Enter Your Code and Execute
SVM = svm.SVC(kernel='rbf')
SVM
```

[42] ✓ 0.0s Python

```
SVC
SVC()
```

```
SVM = SVM.fit(x_train, y_train)
```

[43] ✓ 0.3s Python

Q17) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions`.

```
#Enter Your Code and Execute
predictions = SVM.predict(x_test)
```

[44] ✓ 0.1s Python

```
predictions
```

[45] ✓ 0.0s Python

```
array([0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
```

Q18) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

```
SVM_Accuracy_Score = accuracy_score(y_test, predictions)
SVM_JaccardIndex = jaccard_score(y_test, predictions)
SVM_F1_Score =  f1_score(y_test, predictions)
```

[46] ✓ 0.0s Python

+ Code + Markdown

```
print("Accuracy Score: ", SVM_Accuracy_Score)
print("Jaccard Index: ", SVM_JaccardIndex)
print("F1 Score: ", SVM_F1_Score)
```

[47] ✓ 0.0s Python

```
Accuracy Score:  0.833587786259542
Jaccard Index:  0.47342995169082125
F1 Score:  0.6426229508196721
```

Q19) Show the Accuracy,Jaccard Index,F1-Score and LogLoss in a tabular format using data frame for all of the above models.

*LogLoss is only for Logistic Regression Model

```
Report = pd.DataFrame(data=[['KNN', KNN_Accuracy_Score, KNN_JaccardIndex, KNN_F1_Score],
                            ['Decision Tree', Tree_Accuracy_Score, Tree_JaccardIndex, Tree_F1_Score],
                            ['Logistic Regression', LR_Accuracy_Score, LR_JaccardIndex, LR_F1_Score, LR_Log_Loss],
                            ['SVM', SVM_Accuracy_Score, SVM_JaccardIndex, SVM_F1_Score]],
                      columns=['Model', 'Accuracy Score', 'Jaccard Index', 'F1 Score', 'Log Loss'])
```

[48] ✓ 0.0s Python

+ Code + Markdown

```
Report
```

[49] ✓ 0.0s Python

	Model	Accuracy Score	Jaccard Index	F1 Score	Log Loss
0	KNN	0.760305	0.241546	0.389105	NaN
1	Decision Tree	0.825954	0.495575	0.662722	NaN
2	Logistic Regression	0.844275	0.542601	0.703488	0.372876
3	SVM	0.833588	0.473430	0.642623	NaN