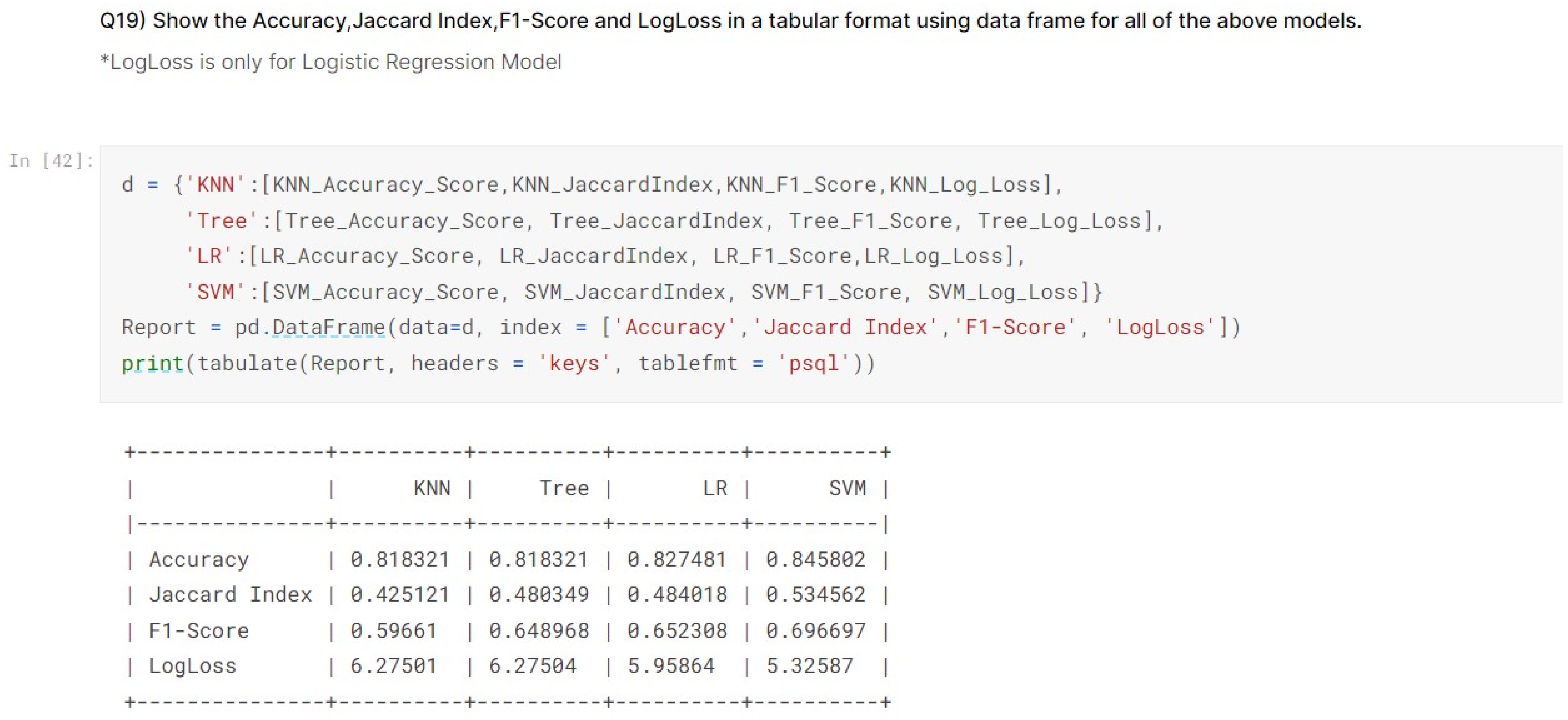


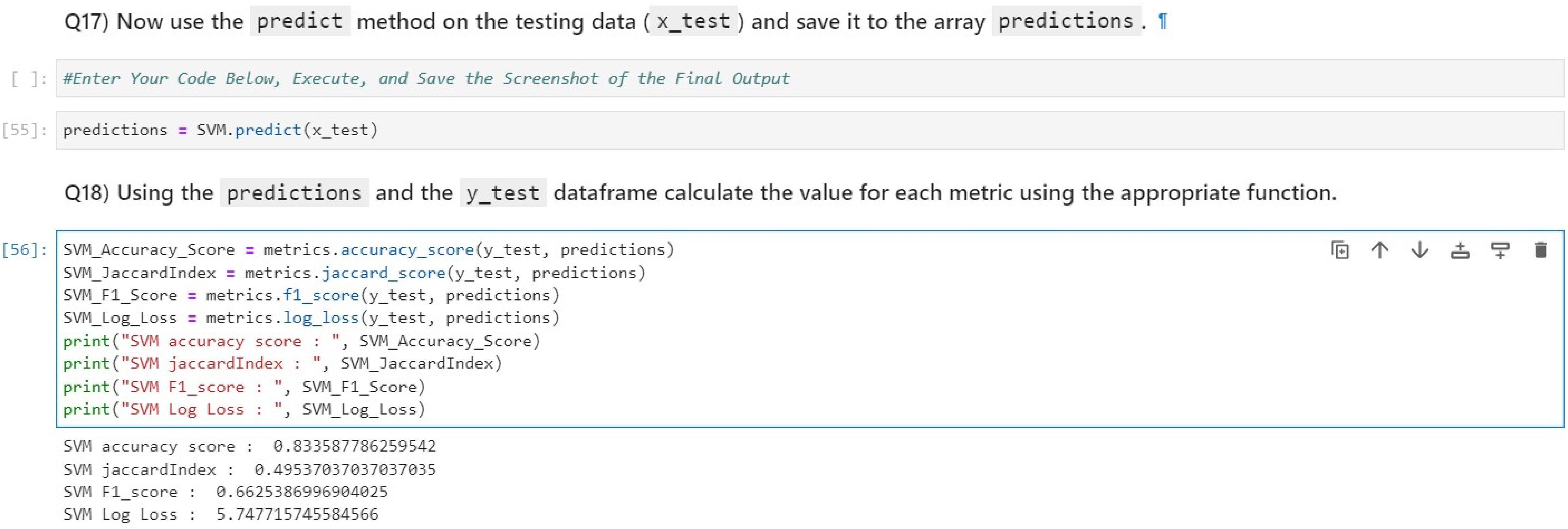
Q19) Show the Accuracy,Jaccard Index,F1-Score and LogLoss in a tabular format using data frame for all of the above models.

*LogLoss is only for Logistic Regression Model

In [42]:

```
d = {'KNN':[KNN_Accuracy_Score,KNN_JaccardIndex,KNN_F1_Score,KNN_Log_Loss],
     'Tree':[Tree_Accuracy_Score, Tree_JaccardIndex, Tree_F1_Score, Tree_Log_Loss],
     'LR':[LR_Accuracy_Score, LR_JaccardIndex, LR_F1_Score,LR_Log_Loss],
     'SVM':[SVM_Accuracy_Score, SVM_JaccardIndex, SVM_F1_Score, SVM_Log_Loss]}
Report = pd.DataFrame(data=d, index = ['Accuracy','Jaccard Index','F1-Score', 'LogLoss'])
print(tabulate(Report, headers = 'keys', tablefmt = 'psql'))
```

```
+---------------+----------+----------+----------+----------+
|               |      KNN |     Tree |       LR |      SVM |
|---------------+----------+----------+----------+----------|
| Accuracy      | 0.818321 | 0.818321 | 0.827481 | 0.845802 |
| Jaccard Index | 0.425121 | 0.480349 | 0.484018 | 0.534562 |
| F1-Score      | 0.59661  | 0.648968 | 0.652308 | 0.696697 |
| LogLoss       | 6.27501  | 6.27504  | 5.95864  | 5.32587  |
+---------------+----------+----------+----------+----------+
```



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

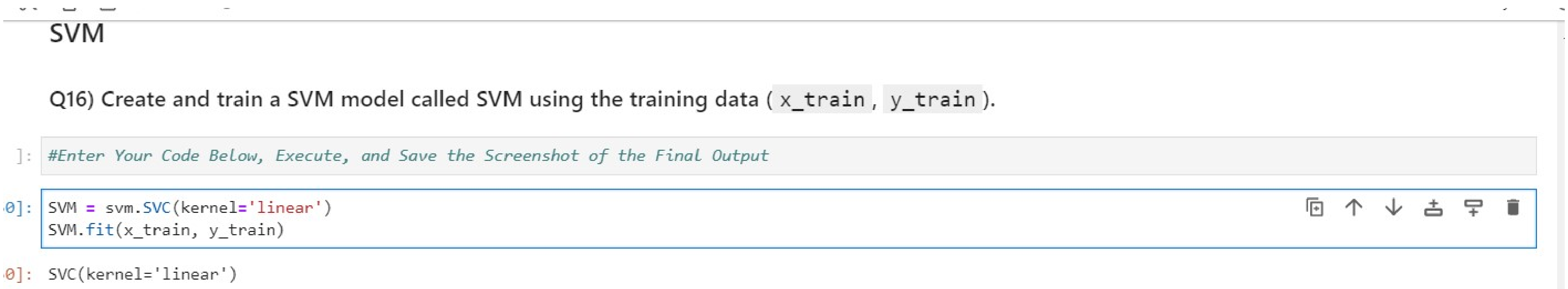
```
[ ]: #Enter Your Code Below, Execute, and Save the Screenshot of the Final Output
```

```
[55]: predictions = SVM.predict(x_test)
```

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

```
[56]: SVM_Accuracy_Score = metrics.accuracy_score(y_test, predictions)
      SVM_JaccardIndex = metrics.jaccard_score(y_test, predictions)
      SVM_F1_Score = metrics.f1_score(y_test, predictions)
      SVM_Log_Loss = metrics.log_loss(y_test, predictions)
      print("SVM accuracy score : ", SVM_Accuracy_Score)
      print("SVM jaccardIndex : ", SVM_JaccardIndex)
      print("SVM F1_score : ", SVM_F1_Score)
      print("SVM Log Loss : ", SVM_Log_Loss)
```

```
SVM accuracy score :  0.833587786259542
SVM jaccardIndex :  0.49537037037037035
SVM F1_score :  0.6625386996904025
SVM Log Loss :  5.747715745584566
```



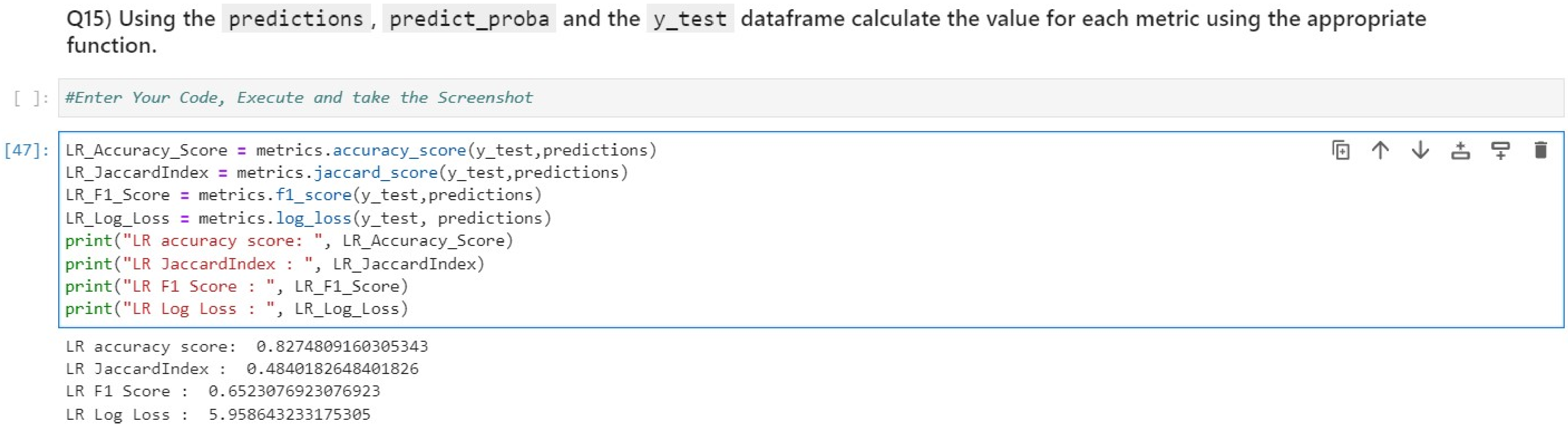
SVM

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

```
]: #Enter Your Code Below, Execute, and Save the Screenshot of the Final Output
```

```
0]: SVM = svm.SVC(kernel='linear')
    SVM.fit(x_train, y_train)
```

```
0]: SVC(kernel='linear')
```

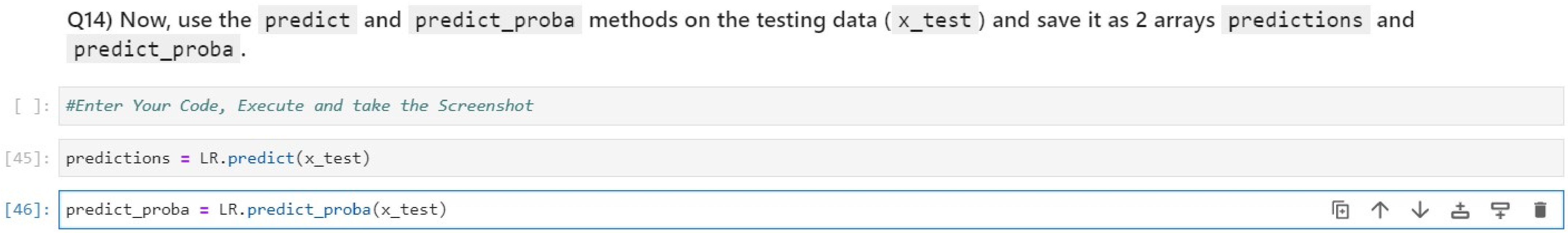


Q15) Using the `predictions`, `predict_proba` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[47]: LR_Accuracy_Score = metrics.accuracy_score(y_test,predictions)
      LR_JaccardIndex = metrics.jaccard_score(y_test,predictions)
      LR_F1_Score = metrics.f1_score(y_test,predictions)
      LR_Log_Loss = metrics.log_loss(y_test, predictions)
      print("LR accuracy score: ", LR_Accuracy_Score)
      print("LR JaccardIndex : ", LR_JaccardIndex)
      print("LR F1 Score : ", LR_F1_Score)
      print("LR Log Loss : ", LR_Log_Loss)
```

```
LR accuracy score:  0.8274809160305343
LR JaccardIndex :  0.4840182648401826
LR F1 Score :  0.6523076923076923
LR Log Loss :  5.958643233175305
```

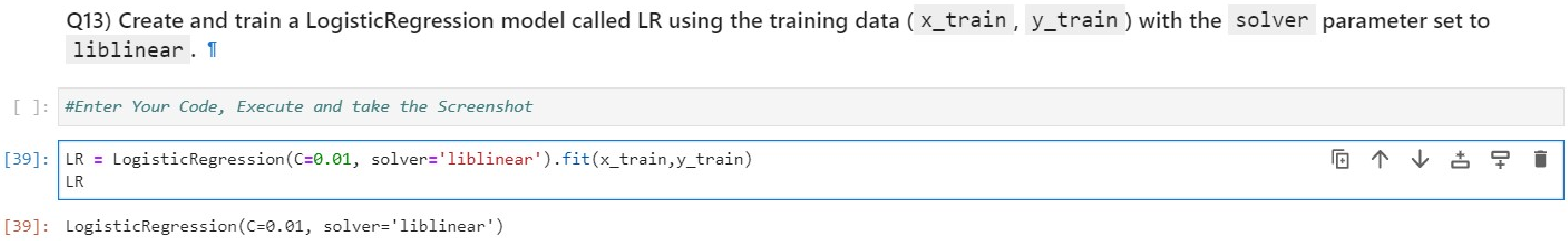


Q14) Now, use the `predict` and `predict_proba` methods on the testing data (`x_test`) and save it as 2 arrays `predictions` and `predict_proba` .

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[45]: predictions = LR.predict(x_test)
```

```
[46]: predict_proba = LR.predict_proba(x_test)
```

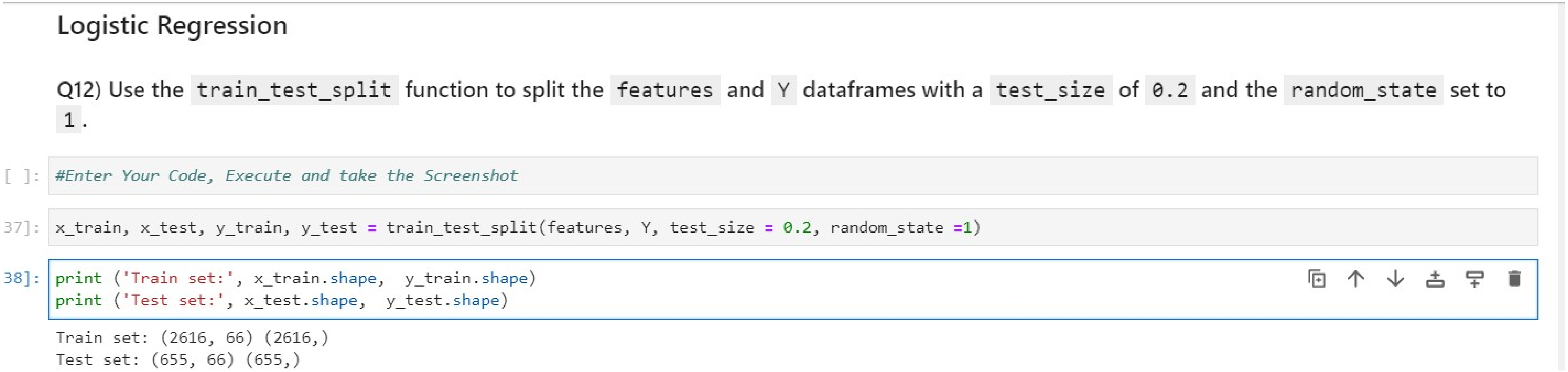


Q13) Create and train a LogisticRegression model called LR using the training data (`x_train` , `y_train`) with the `solver` parameter set to `liblinear` . ¶

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[39]: LR = LogisticRegression(C=0.01, solver='liblinear').fit(x_train,y_train)
      LR
```

```
[39]: LogisticRegression(C=0.01, solver='liblinear')
```



Logistic Regression

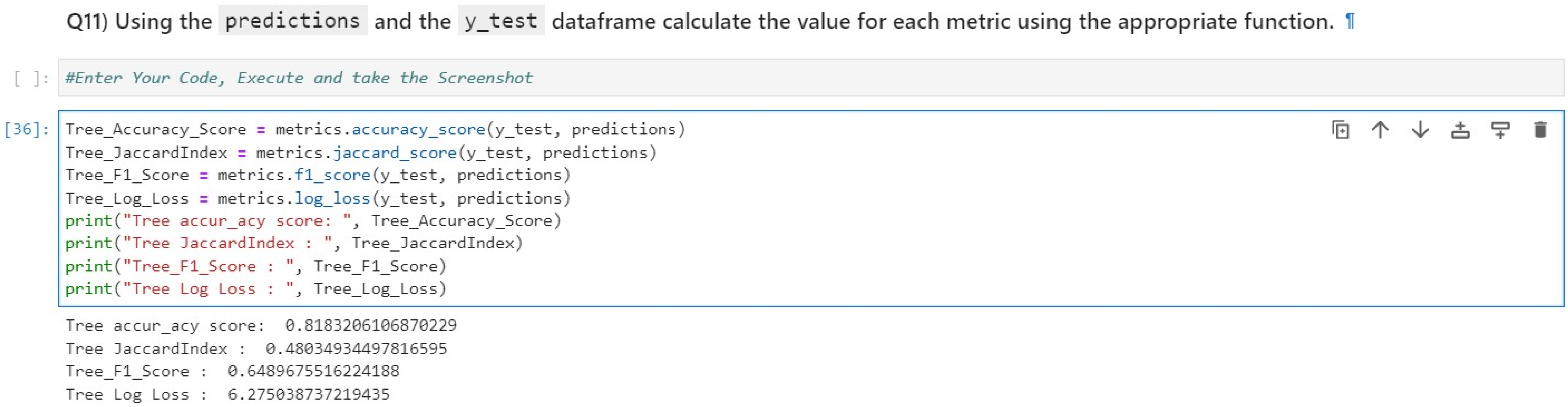
Q12) Use the `train_test_split` function to split the `features` and `Y` dataframes with a `test_size` of `0.2` and the `random_state` set to `1`.

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
37]: x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2, random_state =1)
```

```
38]: print ('Train set:', x_train.shape,  y_train.shape)
     print ('Test set:', x_test.shape,  y_test.shape)
```

```
Train set: (2616, 66) (2616,)
Test set: (655, 66) (655,)
```

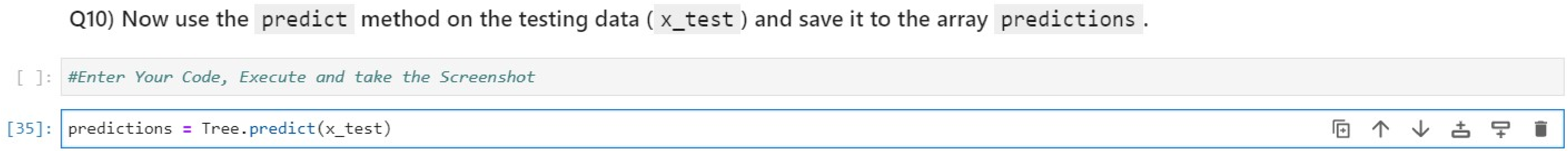


Q11) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function. ¶

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[36]: Tree_Accuracy_Score = metrics.accuracy_score(y_test, predictions)
Tree_JaccardIndex = metrics.jaccard_score(y_test, predictions)
Tree_F1_Score = metrics.f1_score(y_test, predictions)
Tree_Log_Loss = metrics.log_loss(y_test, predictions)
print("Tree accur_acy score: ", Tree_Accuracy_Score)
print("Tree JaccardIndex : ", Tree_JaccardIndex)
print("Tree_F1_Score : ", Tree_F1_Score)
print("Tree Log Loss : ", Tree_Log_Loss)
```

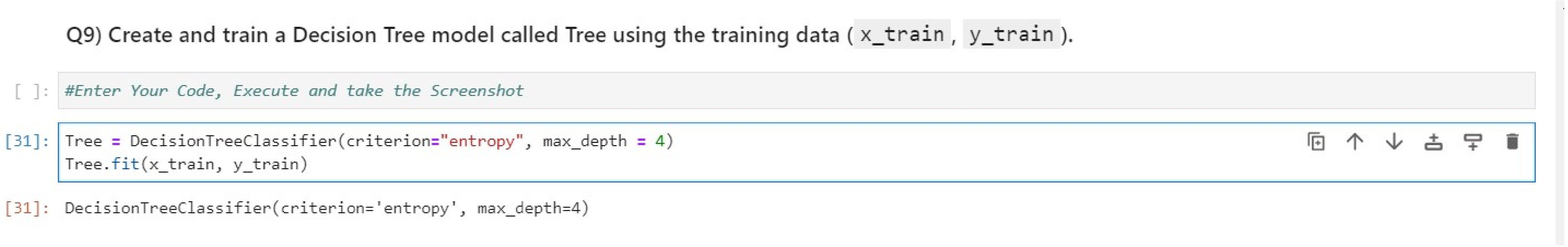
```
Tree accur_acy score:  0.8183206106870229
Tree JaccardIndex :  0.48034934497816595
Tree_F1_Score :  0.6489675516224188
Tree Log Loss :  6.275038737219435
```



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

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[35]: predictions = Tree.predict(x_test)
```

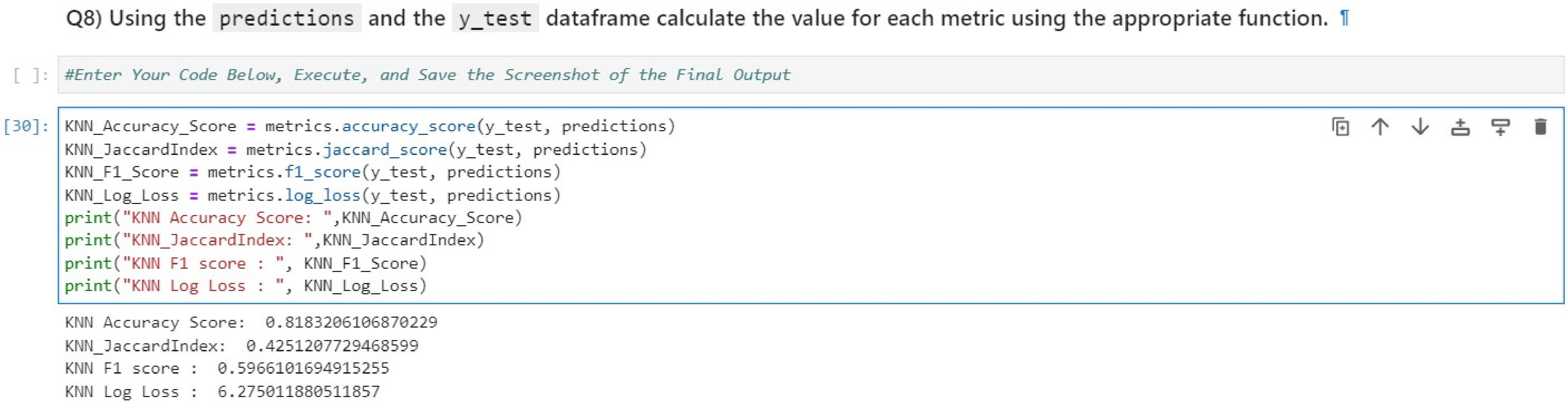


Q9) Create and train a Decision Tree model called Tree using the training data (`x_train`, `y_train`).

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[31]: Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
      Tree.fit(x_train, y_train)
```

```
[31]: DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

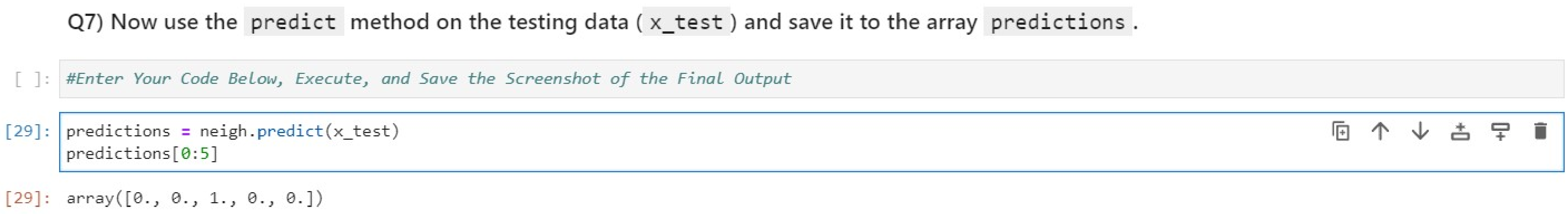


Q8) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function. ¶

```
[ ]: #Enter Your Code Below, Execute, and Save the Screenshot of the Final Output
```

```
[30]: KNN_Accuracy_Score = metrics.accuracy_score(y_test, predictions)
      KNN_JaccardIndex = metrics.jaccard_score(y_test, predictions)
      KNN_F1_Score = metrics.f1_score(y_test, predictions)
      KNN_Log_Loss = metrics.log_loss(y_test, predictions)
      print("KNN Accuracy Score: ",KNN_Accuracy_Score)
      print("KNN_JaccardIndex: ",KNN_JaccardIndex)
      print("KNN F1 score : ", KNN_F1_Score)
      print("KNN Log Loss : ", KNN_Log_Loss)
```

```
KNN Accuracy Score:  0.8183206106870229
KNN_JaccardIndex:  0.4251207729468599
KNN F1 score :  0.5966101694915255
KNN Log Loss :  6.275011880511857
```

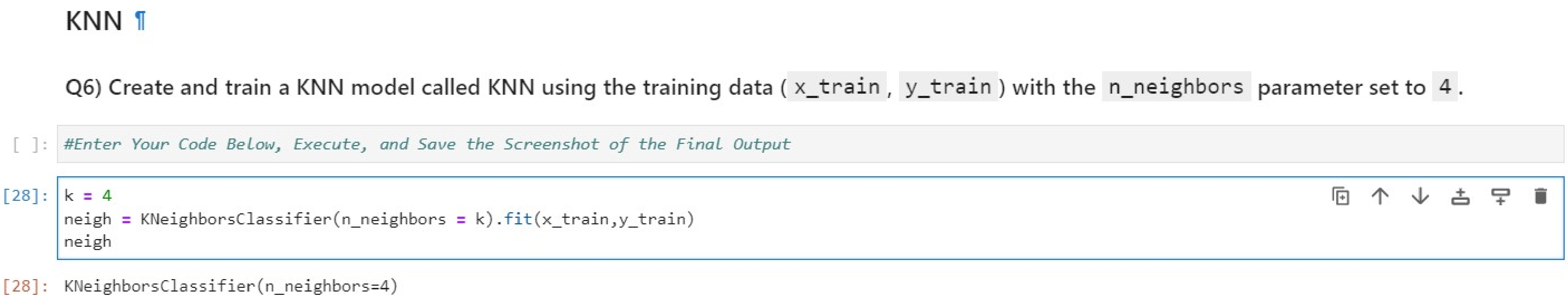


Q7) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions` .

```
[ ]: #Enter Your Code Below, Execute, and Save the Screenshot of the Final Output
```

```
[29]: predictions = neigh.predict(x_test)
      predictions[0:5]
```

```
[29]: array([0., 0., 1., 0., 0.])
```



KNN ¶

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 Below, Execute, and Save the Screenshot of the Final Output
```

```
[28]: k = 4
      neigh = KNeighborsClassifier(n_neighbors = k).fit(x_train,y_train)
      neigh
```

```
[28]: KNeighborsClassifier(n_neighbors=4)
```

In [14]:

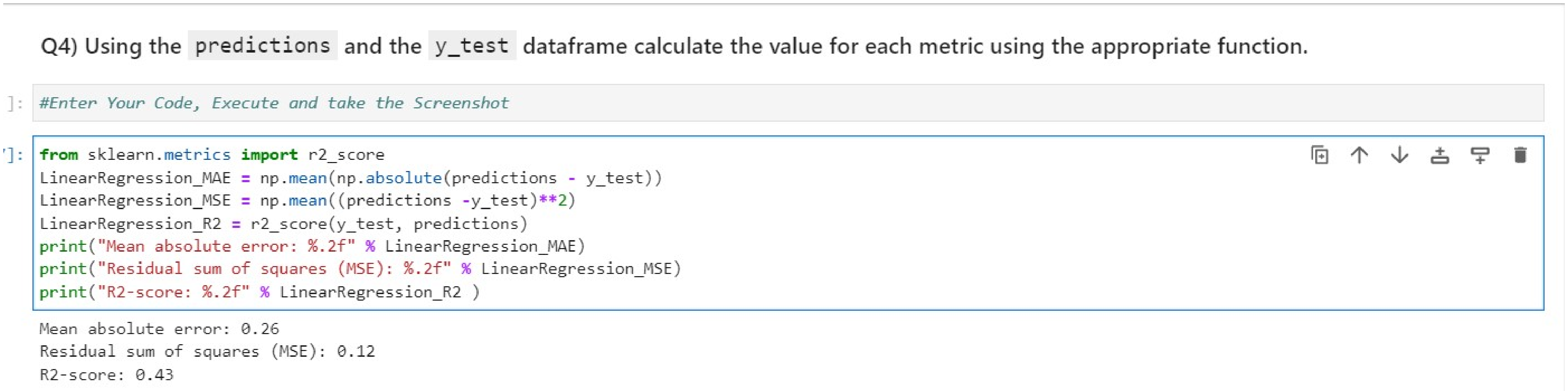
```
dict = {'error_type':['LinearRegression_MAE','LinearRegression_MSE','LinearRegression_R2'],

        'value':[LinearRegression_MAE,LinearRegression_MSE,LinearRegression_R2]}
```

In [15]:

```
from tabulate import tabulate
Report = pd.DataFrame(dict)
print(tabulate(Report, headers = 'keys', tablefmt = 'psql'))
```

```
+----+---------------------+----------+
|    | error_type          |    value |
|----+---------------------+----------|
|  0 | LinearRegression_MAE | 0.256318 |
|  1 | LinearRegression_MSE | 0.115721 |
|  2 | LinearRegression_R2  | 0.427132 |
+----+---------------------+----------+
```

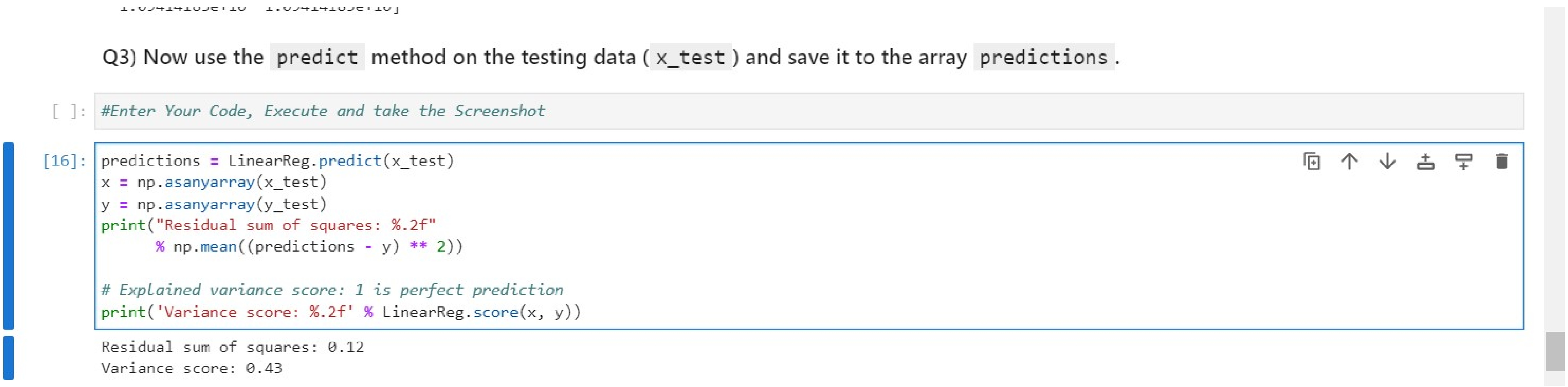


Q4) Using the `predictions` and the `y_test` dataframe calculate the value for each metric using the appropriate function.

```
#Enter Your Code, Execute and take the Screenshot
```

```
from sklearn.metrics import r2_score
LinearRegression_MAE = np.mean(np.absolute(predictions - y_test))
LinearRegression_MSE = np.mean((predictions -y_test)**2)
LinearRegression_R2 = r2_score(y_test, predictions)
print("Mean absolute error: %.2f" % LinearRegression_MAE)
print("Residual sum of squares (MSE): %.2f" % LinearRegression_MSE)
print("R2-score: %.2f" % LinearRegression_R2 )
```

```
Mean absolute error: 0.26
Residual sum of squares (MSE): 0.12
R2-score: 0.43
```



```
 1.09414185e+10  1.09414185e+10]
```

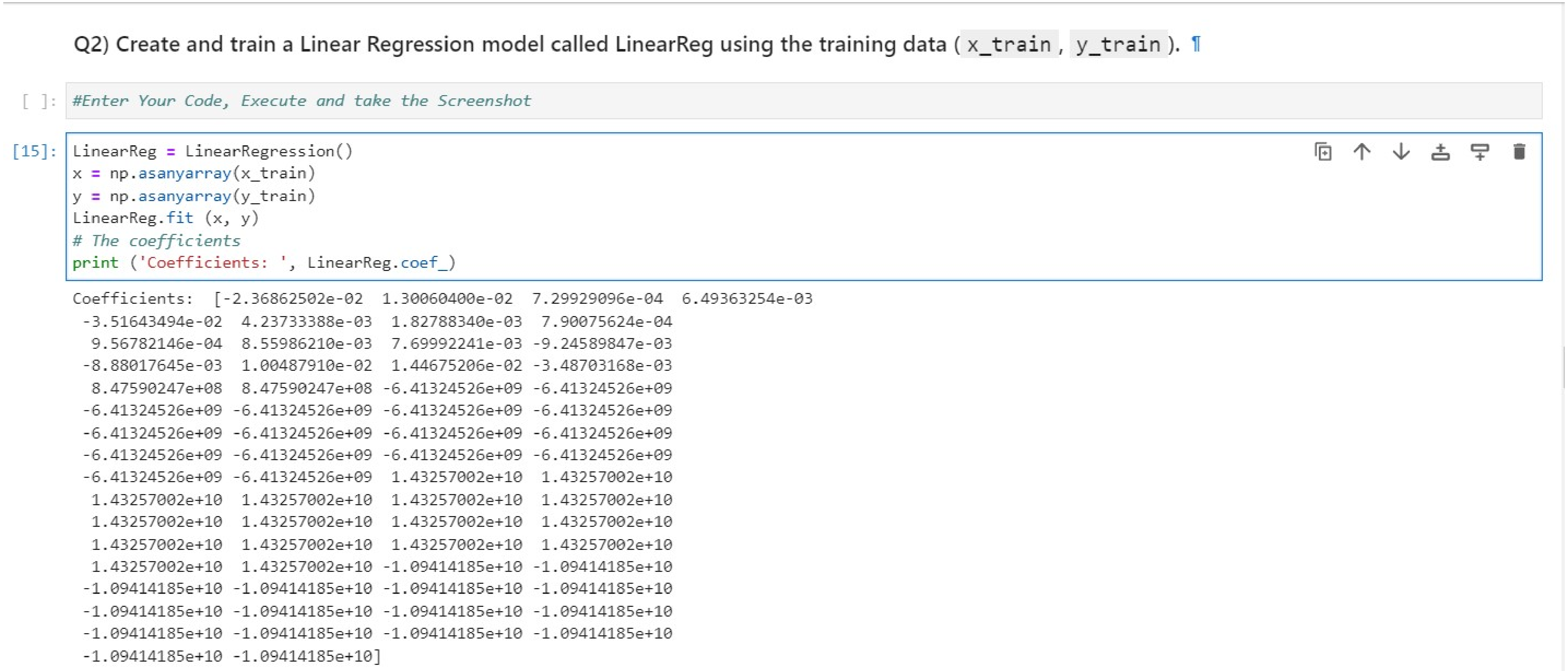
Q3) Now use the `predict` method on the testing data (`x_test`) and save it to the array `predictions` .

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[16]: predictions = LinearReg.predict(x_test)
      x = np.asanyarray(x_test)
      y = np.asanyarray(y_test)
      print("Residual sum of squares: %.2f"
            % np.mean((predictions - y) ** 2))

      # Explained variance score: 1 is perfect prediction
      print('Variance score: %.2f' % LinearReg.score(x, y))
```

```
Residual sum of squares: 0.12
Variance score: 0.43
```

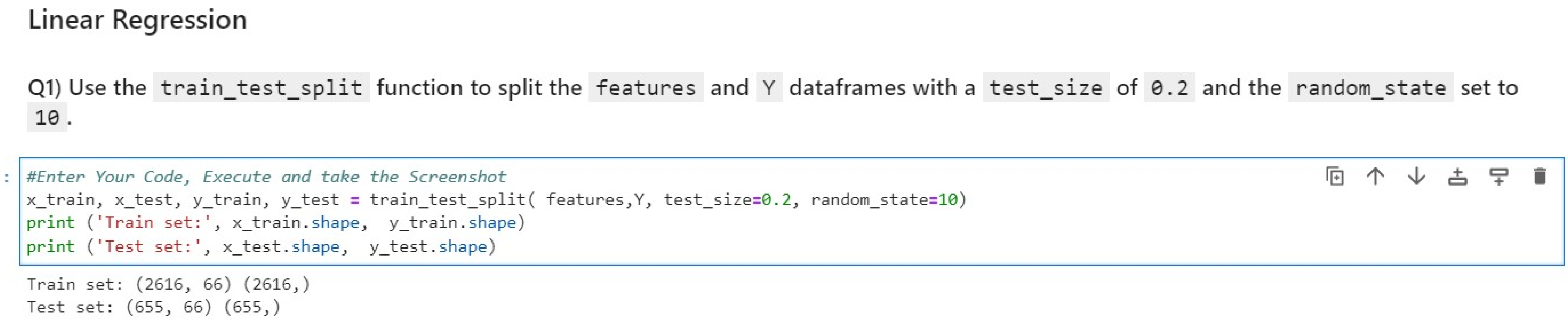


Q2) Create and train a Linear Regression model called LinearReg using the training data (`x_train` , `y_train`). ¶

```
[ ]: #Enter Your Code, Execute and take the Screenshot
```

```
[15]: LinearReg = LinearRegression()
      x = np.asanyarray(x_train)
      y = np.asanyarray(y_train)
      LinearReg.fit (x, y)
      # The coefficients
      print ('Coefficients: ', LinearReg.coef_)
```

```
Coefficients:  [-2.36862502e-02  1.30060400e-02  7.29929096e-04  6.49363254e-03
 -3.51643494e-02  4.23733388e-03  1.82788340e-03  7.90075624e-04
  9.56782146e-04  8.55986210e-03  7.69992241e-03 -9.24589847e-03
 -8.88017645e-03  1.00487910e-02  1.44675206e-02 -3.48703168e-03
  8.47590247e+08  8.47590247e+08 -6.41324526e+09 -6.41324526e+09
 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09
 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09
 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09 -6.41324526e+09
 -6.41324526e+09 -6.41324526e+09  1.43257002e+10  1.43257002e+10
  1.43257002e+10  1.43257002e+10  1.43257002e+10  1.43257002e+10
  1.43257002e+10  1.43257002e+10  1.43257002e+10  1.43257002e+10
  1.43257002e+10  1.43257002e+10  1.43257002e+10  1.43257002e+10
  1.43257002e+10  1.43257002e+10 -1.09414185e+10 -1.09414185e+10
 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10
 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10
 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10 -1.09414185e+10
 -1.09414185e+10 -1.09414185e+10]
```



Linear Regression

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, Execute and take the Screenshot
x_train, x_test, y_train, y_test = train_test_split( features,Y, test_size=0.2, random_state=10)
print ('Train set:', x_train.shape,  y_train.shape)
print ('Test set:', x_test.shape,  y_test.shape)
```

```
Train set: (2616, 66) (2616,)
Test set: (655, 66) (655,)
```