# Homework 5 - Data Science for Everyone

May 1, 2022

**Name:** Suvir Wadhwa, **N-Number:** N16395336

[66]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
```

[67]:
```
data = pd.read_csv("fifa22.csv") #Load fifa data using pandas
data.head()
```

[67]:
```
                             name  rank gender  wage_eur   log_wage position  \
0  Lionel Andrés Messi Cuccittini    93      M  320000.0  12.676076       RW
1      Lucia Roberta Tough Bronze    92      F       NaN        NaN      NaN
2                Vivianne Miedema    92      F       NaN        NaN      NaN
3     Wéndèleine Thérèse Renard    92      F       NaN        NaN      NaN
4              Robert Lewandowski    92      M  270000.0  12.506177       ST

  nationality                club                  league preferred_foot  \
0   Argentina  Paris Saint-Germain        French Ligue 1           Left
1     England                 NaN                     NaN          Right
2 Netherlands                 NaN                     NaN          Right
3      France                 NaN                     NaN          Right
4      Poland    FC Bayern München  German 1. Bundesliga          Right

  shooting  passing  dribbling  defending  attacking  skill  movement  power  \
0     92.0     91.0       95.0  26.333333       85.8   94.0      90.2   77.8
1     61.0     70.0       81.0  89.000000       69.0   62.2      84.2   78.8
2     93.0     75.0       88.0  25.000000       86.0   79.0      80.6   84.0
3     70.0     62.0       73.0  91.333333       62.6   67.8      64.0   82.4
4     92.0     79.0       86.0  32.000000       86.0   81.4      81.6   84.8

  mentality  goalkeeping
0 73.833333         10.8
1 69.166667         12.6
2 70.833333         15.6
3 73.500000         12.8
4 80.666667         10.2
```

(b) The unit of analysis in this study is the skill level of players different football players.

[68]:
```
data.shape
```

[68]: (19630, 20)

(c) The data has 19630 observations and 20 different features.

[69]:
```
(data['gender'] == "M").sum() #Count for number of Male characters
```

[69]: 19239

[70]:
```
(data['gender'] == "F").sum() #Count for number of Female characters
```

[70]: 391

(e) Yes, to some degree, this data set covers the skill levels and standards of most professional level soccer levels in the world. This is because it covers players from almost every professional league and it also attributes their different skill levels, hence, representing the players to a higher degree.

[71]:
```
data = data.dropna(subset=['passing']) #Drop all of the NA rows in the passing␣
↪column
data.shape
```

[71]: (17450, 20)

**Question 2**

[72]:
```
import statsmodels.formula.api as smf
```

[73]:
```
results = smf.ols('rank ~ passing + attacking + defending + skill', data=data).
↪fit()     # multiple linear regression
results.summary()
```

[73]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   rank   R-squared:                       0.705
Model:                            OLS   Adj. R-squared:                  0.705
Method:                 Least Squares   F-statistic:                 1.044e+04
Date:                Sun, 01 May 2022   Prob (F-statistic):               0.00
Time:                        21:51:06   Log-Likelihood:                -47856.
No. Observations:               17450   AIC:                         9.572e+04
Df Residuals:                   17445   BIC:                         9.576e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
Intercept     25.3278      0.203    124.785      0.000      24.930      25.726
passing       -0.0247      0.010     -2.425      0.015      -0.045      -0.005
attacking      0.6109      0.006     94.005      0.000       0.598       0.624
defending      0.1719      0.002     84.413      0.000       0.168       0.176
skill          0.0066      0.009      0.730      0.465      -0.011       0.024
==============================================================================
Omnibus:                      171.799   Durbin-Watson:                   1.342
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              178.339
Skew:                           0.234   Prob(JB):                     1.88e-39
Kurtosis:                       3.163   Cond. No.                         790.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

(b) 70.5% of the variance in rank is explained by our features

(c) Attacking and defending are the 2 features that are significant at the 1% level.

(d) A 1-unit increase in skill is associated with a 0.0066 increase in ranking holding everything else constant.

**Question 3**

(a) Based on the statsmodels output from Q2, the four features could do a pretty good job at predicting the rank for out-of-sample data. This is because the variance in data caused by the IVs is more that 70% and the standard errors for each IV are minimal.

[74]:
```python
# SciKit Learn packages
from sklearn.model_selection import train_test_split # for splitting data
from sklearn.preprocessing import StandardScaler     # feature scaling
from sklearn import metrics                          # for evaluation metrics

from sklearn.linear_model import LinearRegression    # linear regression
from sklearn.neighbors import KNeighborsClassifier   # knn
```

[75]:
```python
X = data[['passing', 'attacking', 'defending', 'skill']]
Y = data[['rank']]
X.head()
```

[75]:

| | passing | attacking | defending | skill |
|---|---|---|---|---|
| 0 | 91.0 | 85.8 | 26.333333 | 94.0 |
| 1 | 70.0 | 69.0 | 89.000000 | 62.2 |
| 2 | 75.0 | 86.0 | 25.000000 | 79.0 |
| 3 | 62.0 | 62.6 | 91.333333 | 67.8 |
| 4 | 79.0 | 86.0 | 32.000000 | 81.4 |

[76]:
```
Y.head()
```

[76]:

| | rank |
|---|---|
| 0 | 93 |
| 1 | 92 |
| 2 | 92 |
| 3 | 92 |
| 4 | 92 |

[77]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25,␣
↪random_state = 123)
```

[78]:
```
x_train.head()
```

[78]:

| | passing | attacking | defending | skill |
|---|---|---|---|---|
| 17226 | 52.0 | 48.0 | 59.333333 | 53.2 |
| 13548 | 48.0 | 55.0 | 12.666667 | 54.0 |
| 17874 | 59.0 | 46.2 | 58.000000 | 57.8 |
| 19599 | 47.0 | 40.6 | 46.666667 | 40.0 |
| 15629 | 49.0 | 51.8 | 25.666667 | 49.6 |

[79]:
```
# train our model
trained = LinearRegression().fit(x_train, y=y_train)

print('Intercept', trained.intercept_)        # Print the intercept
print('Coefficient', trained.coef_)           # Print the coefficient
```

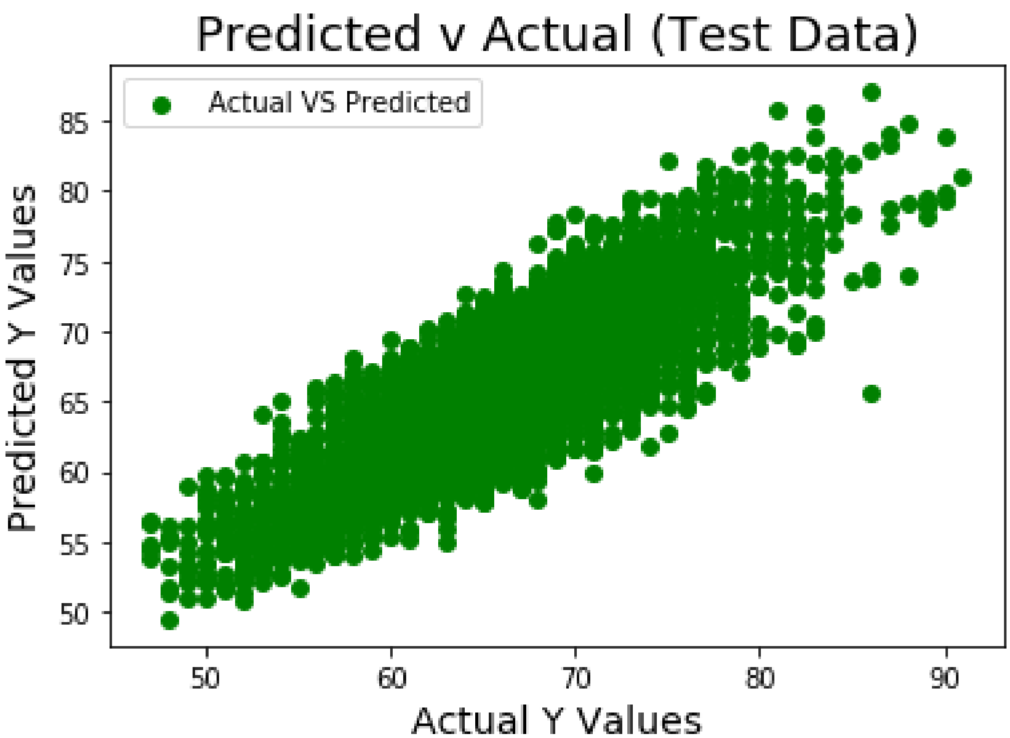
```
Intercept [25.16773306]
Coefficient [[-0.02444506  0.61230756  0.17314968  0.00612364]]
```

(e) The coefficent for attacking in Q2 was 0.6109. While the coefficient for attacking in this question is 0.6123. The coefficient hasn't changed too much between the two questions. It has increased by around 0.0014.

[80]:
```
y_pred = trained.predict(x_test)
y_pred[:3]
```

[80]:
```
array([[64.57617047],
       [72.78035994],
       [70.46341746]])
```

[81]:
```
import matplotlib.pyplot as plt
plt.scatter(y_test, y_pred, color = 'green', label='Actual VS Predicted')
plt.xlabel('Actual Y Values', size=14)
plt.ylabel('Predicted Y Values', size=14)
plt.title('Predicted v Actual (Test Data)', size=18)
plt.legend()
plt.show()
```




[82]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred)))
print('R-squared', metrics.r2_score(y_test, y_pred))
```

```
Mean Absolute Error: 2.9749813133233594
Mean Squared Error: 14.021749364787894
Root Mean Squared Error: 3.744562639987198
R-squared 0.6986015036415898
```

(h) It is the square root of the average error of the model. It enables us to identify outliers more easily.

(i) I believe the model did an average job in predicting the rank of the players. Its R-squared value was 0.69 indicating the fact that it is not 100% accurate

**Question 4**

[83]:
```
data['preferred_foot'].value_counts()
```

[83]:
```
Right    13044
Left      4406
Name: preferred_foot, dtype: int64
```

[84]:
```
pref = (13044/(13044 + 4406)) * 100
print("{:2f} percent of players prefer their right foot.".format(pref))
```

```
74.750716 percent of players prefer their right foot.
```

[85]:
```
X2 = data[['shooting', 'passing', 'dribbling', 'defending', 'attacking',
           'skill', 'movement', 'power', 'mentality', 'goalkeeping']]
X2.head()
```

[85]:

| | shooting | passing | dribbling | defending | attacking | skill | movement | power | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 92.0 | 91.0 | 95.0 | 26.333333 | 85.8 | 94.0 | 90.2 | 77.8 | |
| 1 | 61.0 | 70.0 | 81.0 | 89.000000 | 69.0 | 62.2 | 84.2 | 78.8 | |
| 2 | 93.0 | 75.0 | 88.0 | 25.000000 | 86.0 | 79.0 | 80.6 | 84.0 | |
| 3 | 70.0 | 62.0 | 73.0 | 91.333333 | 62.6 | 67.8 | 64.0 | 82.4 | |
| 4 | 92.0 | 79.0 | 86.0 | 32.000000 | 86.0 | 81.4 | 81.6 | 84.8 | |

| | mentality | goalkeeping |
|---|---|---|
| 0 | 73.833333 | 10.8 |
| 1 | 69.166667 | 12.6 |
| 2 | 70.833333 | 15.6 |
| 3 | 73.500000 | 12.8 |
| 4 | 80.666667 | 10.2 |

[86]:
```
# rescale IVs
scaler = StandardScaler().fit(X2) #Scale the data
x_scaled = scaler.transform(X2)
x_scaled = pd.DataFrame(x_scaled, columns= X2.columns)
x_scaled.head()
```

[86]:

| | shooting | passing | dribbling | defending | attacking | skill | movement | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 2.784312 | 3.296642 | 3.315358 | -1.393049 | 3.400164 | 3.548580 | 2.774640 | |
| 1 | 0.597719 | 1.229719 | 1.876719 | 2.131667 | 1.593417 | 0.598209 | 2.072809 | |
| 2 | 2.854847 | 1.721843 | 2.596039 | -1.468043 | 3.421673 | 2.156896 | 1.651710 | |
| 3 | 1.232536 | 0.442320 | 1.054640 | 2.262906 | 0.905132 | 1.117771 | -0.290023 | |
| 4 | 2.784312 | 2.115543 | 2.390519 | -1.074325 | 3.421673 | 2.379565 | 1.768682 | |

| | power | mentality | goalkeeping |
|---|---|---|---|
| 0 | 1.944282 | 2.180614 | 0.281676 |
| 1 | 2.066501 | 1.623697 | 1.481100 |
| 2 | 2.702042 | 1.822596 | 3.480141 |
| 3 | 2.506491 | 2.140834 | 1.614369 |
| 4 | 2.799817 | 2.996099 | -0.118132 |

[87]:
```
Y2 = data['preferred_foot']
x2_train, x2_test, y2_train, y2_test = train_test_split(x_scaled, Y2,
                                                    test_size=0.30,␣
 ↪random_state = 456)
```

```
x2_train.head()
```

[87]:

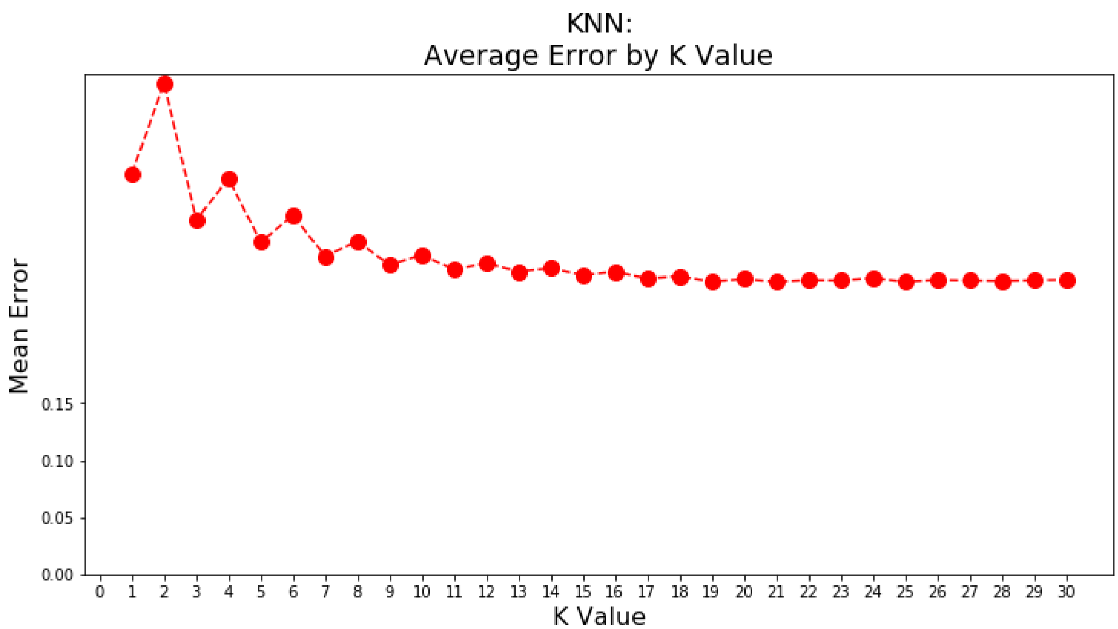
```
        shooting   passing  dribbling  defending  attacking     skill  \
15473 -2.012086 -1.427753  -1.514357   0.444303  -1.826497 -1.572819
9895  -0.460310  0.343895   0.746361   0.425554  -0.277857  0.616765
12369  0.315578  0.147045   0.129801  -0.886840   0.087794  0.004424
4032   1.655747  2.017118   0.951880  -0.286888   1.206257  2.064117
3573  -1.871015  0.245470   0.027041   1.137997  -0.578981 -0.181134

       movement     power  mentality  goalkeeping
15473 -0.968460 -1.013424  -1.558684     0.548214
9895   0.692541 -0.646766  -0.305621    -0.251402
12369 -0.056079 -0.402328  -0.365291    -1.850634
4032   0.552174  0.599870   0.788322     1.614369
3573  -0.453784 -0.744542   0.450194     0.281676
```

[88]:

```
error = list()    # list to store errors

# "sweep" the parameter space for k between 1 and 30
for k in range(1, 31):
    knn = KNeighborsClassifier(n_neighbors= k) # init our knn object
    knn.fit(x2_train, y2_train)     # fit the model
    y2_pred = knn.predict(x2_test) # get our predictions
    error.append(np.mean(y2_pred != y2_test)) # record the mean error
```

[89]:

```
# plot error
plt.figure(figsize=(12, 6))

plt.plot(range(1, len(error)+1), error, color='red', linestyle='dashed',␣
↪marker='o',
         markersize=10)

plt.yticks(np.arange(0, 0.2, .05)) # start y ticks at 0
plt.xticks(np.arange(0, 31, 1))  # integer x ticks

# title & label axes
plt.title('KNN:\nAverage Error by K Value', size=18)
plt.xlabel('K Value', size=16)
plt.ylabel('Mean Error', size=16)
plt.show()
```




[90]:
```
# find minimum error / k that goes with it
mins = np.where(error==min(error))[0]

print('Minimum error found at:')
for i in mins:
    print('k = {}, error = {:.2}'.format(i+1, error[i]))
```

```
Minimum error found at:
k = 21, error = 0.26
k = 25, error = 0.26
```

[91]:
```
neighbors = 21 # Lowest error is at k = 21
classifier = KNeighborsClassifier(n_neighbors= neighbors)
classifier.fit(x2_train, y2_train)
y2_pred = classifier.predict(x2_test)                  # make predictions on␣
↪our test data
y2_pred[:3] #first 3 predictions for preffered foot
```

[91]:
```
array(['Right', 'Right', 'Right'], dtype=object)
```

[92]:
```
print(metrics.confusion_matrix(y2_test, y2_pred))
```

```
[[  67 1259]
 [  85 3824]]
```

(h) 85 players actually prefer to use their left foot but were predicted to use their right.

[93]:
```
print(metrics.classification_report(y2_test, y2_pred))
```

```
              precision    recall  f1-score   support

        Left       0.44      0.05      0.09      1326
       Right       0.75      0.98      0.85      3909

    accuracy                           0.74      5235
   macro avg       0.60      0.51      0.47      5235
weighted avg       0.67      0.74      0.66      5235
```

(i) The recall for the classification 'left' is 0.05, suggesting that not too many true values were found for left.

(j) The model did a bad job at preddicting the player's preffered foot.

**Question 5**

[94]:
```
x_scaled.head()
```

[94]:

| | shooting | passing | dribbling | defending | attacking | skill | movement |
|---|---|---|---|---|---|---|---|
| 0 | 2.784312 | 3.296642 | 3.315358 | -1.393049 | 3.400164 | 3.548580 | 2.774640 |
| 1 | 0.597719 | 1.229719 | 1.876719 | 2.131667 | 1.593417 | 0.598209 | 2.072809 |
| 2 | 2.854847 | 1.721843 | 2.596039 | -1.468043 | 3.421673 | 2.156896 | 1.651710 |
| 3 | 1.232536 | 0.442320 | 1.054640 | 2.262906 | 0.905132 | 1.117771 | -0.290023 |
| 4 | 2.784312 | 2.115543 | 2.390519 | -1.074325 | 3.421673 | 2.379565 | 1.768682 |

| | power | mentality | goalkeeping |
|---|---|---|---|
| 0 | 1.944282 | 2.180614 | 0.281676 |
| 1 | 2.066501 | 1.623697 | 1.481100 |
| 2 | 2.702042 | 1.822596 | 3.480141 |
| 3 | 2.506491 | 2.140834 | 1.614369 |
| 4 | 2.799817 | 2.996099 | -0.118132 |

[95]:
```
k_Xdata = pd.DataFrame.sample(x_scaled, n = 5000, random_state = 2022)
k_Xdata.head()
```

[95]:

| | shooting | passing | dribbling | defending | attacking | skill |
|---|---|---|---|---|---|---|
| 291 | 1.373606 | 2.115543 | 1.465680 | 1.550464 | 1.830015 | 1.748668 |
| 501 | 1.937889 | 0.934444 | 1.876719 | -0.849343 | 1.959068 | 1.377552 |
| 8871 | 1.020930 | -0.246654 | -0.795038 | -0.736852 | 1.120221 | -0.979033 |
| 12793 | 0.456648 | -0.345079 | 0.129801 | -1.580534 | 0.173830 | -0.552250 |
| 7256 | -1.377269 | -0.541929 | -1.206077 | 0.763027 | -0.600490 | -1.591375 |

| | movement | power | mentality | goalkeeping |
|---|---|---|---|---|
| 291 | 0.037498 | 1.944282 | 2.180614 | 0.948023 |
| 501 | 2.049414 | 1.870951 | 1.404908 | 0.148406 |
| 8871 | -1.576714 | 0.990972 | 0.310965 | 0.281676 |

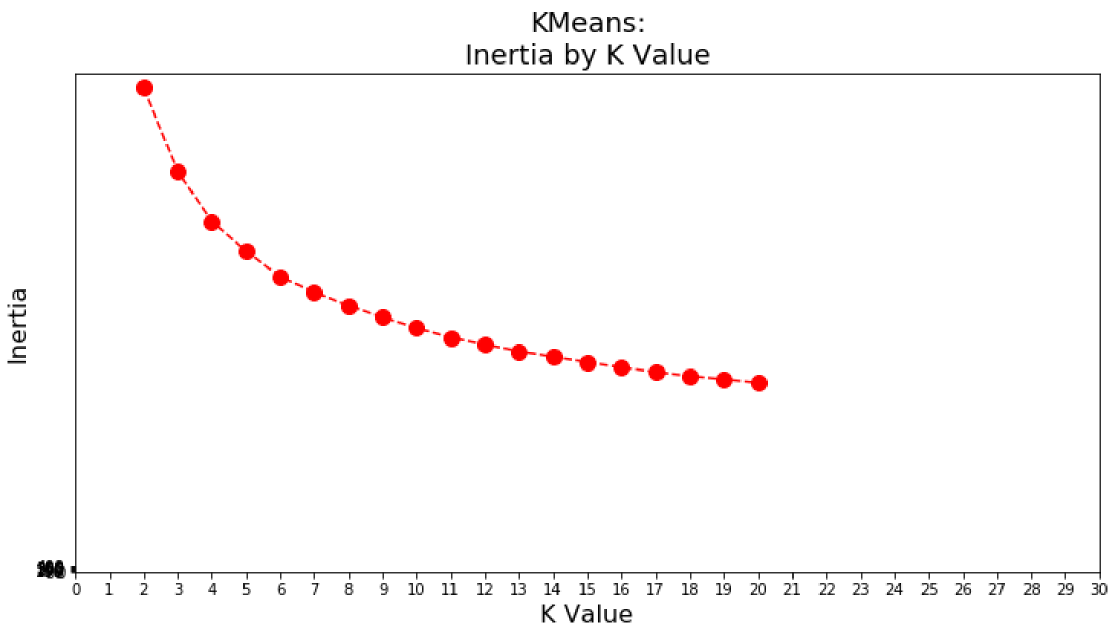
```
12793  1.090245  1.039860  -0.703419    -1.051018
7256  -0.430389  0.013218  -0.206172     0.814753
```

[96]:

```
# calculating error and silhouette in the same loop
from sklearn.cluster import KMeans
error = list()  # to save error/inertia
sil = list()  # to save sillhoute score

for k in range(2,21):
    kmeans = KMeans(n_clusters = k, random_state=789) # init k-means object
    kmeans.fit(k_Xdata)  # run k-means!

    error.append(kmeans.inertia_) # save sum of squared error (SSE)
    score = metrics.silhouette_score(k_Xdata, kmeans.labels_) # calc silhouette␣
↪score
    sil.append(score) # save score
    #calculating error and silhouette in the same loop
```

[97]:

```
# plot SSE
plt.figure(figsize=(12, 6))

plt.plot(range(2, len(error)+2), error, color='red', linestyle='dashed',␣
↪marker='o',
         markersize=10)

plt.yticks(np.arange(0, 401, 50)) # start y ticks at 0
plt.xticks(np.arange(0, 31, 1))  # integer x ticks

# title & label axes
plt.title('KMeans:\nInertia by K Value', size=18)
plt.xlabel('K Value', size=16)
plt.ylabel('Inertia', size=16)
plt.show()
```




[98]:

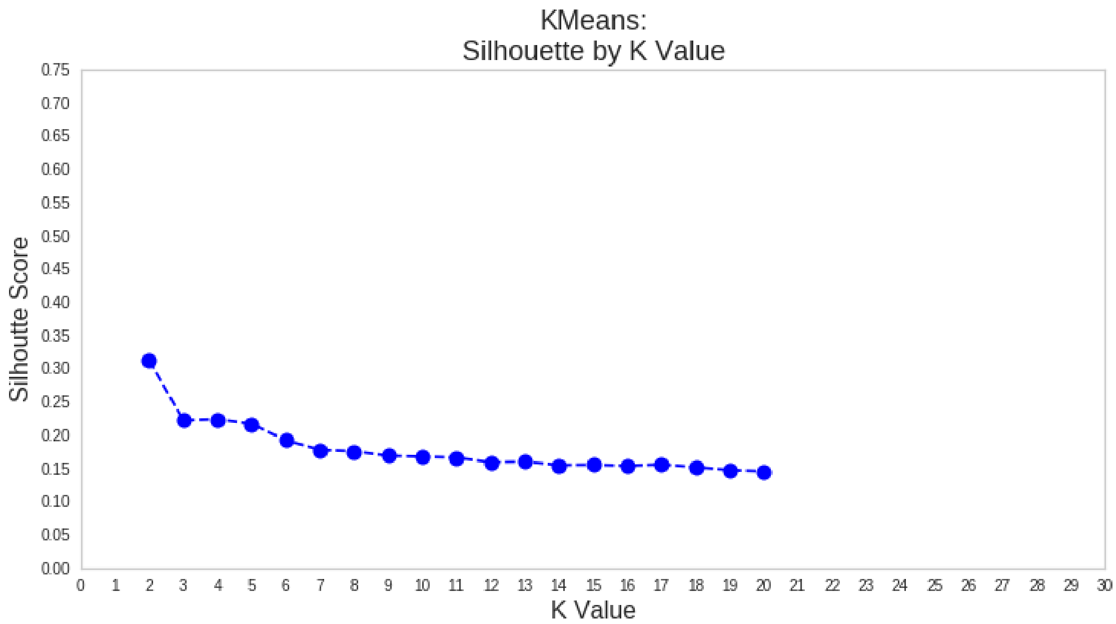
```
# additional metrics
from kneed import KneeLocator # for KMeans, elbow method
from yellowbrick.cluster import SilhouetteVisualizer # for KMeans, silhouette␣
↪scores
# find elbow
kl = KneeLocator(range(2, 21), error,
                  curve='convex', direction='decreasing')
print(kl.elbow)
```

6

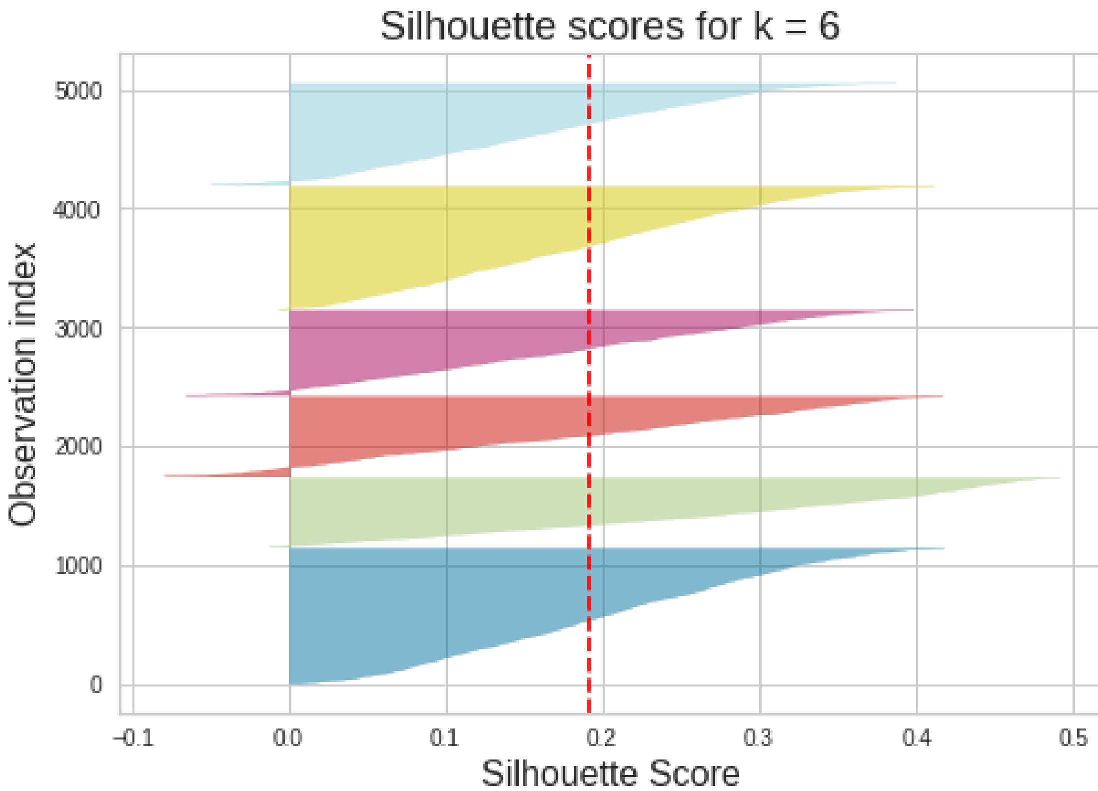
[99]:

```
# plot Silhouette
plt.figure(figsize=(12, 6))
plt.grid(False)

plt.plot(range(2, len(sil)+2), sil, color='blue', linestyle='dashed',␣
↪marker='o',
         markersize=10)

plt.yticks(np.arange(0, .76, .05)) # start y ticks at 0
plt.xticks(np.arange(0, 31, 1))  # integer x ticks

# title & label axes
plt.title('KMeans:\nSilhouette by K Value', size=18)
plt.xlabel('K Value', size=16)
plt.ylabel('Silhoutte Score', size=16)
```

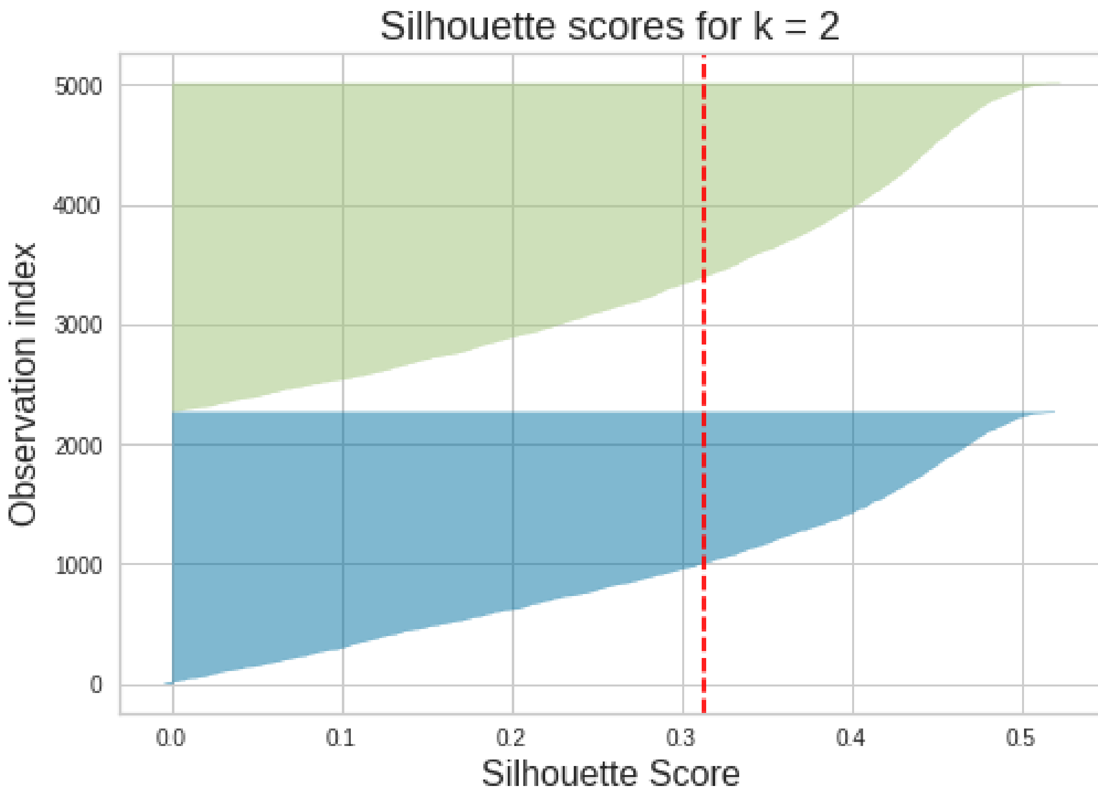
```
plt.show()
```




[100]:
```
# look at the silhouette plot of this

###### set value of k
k = 6

# Create KMeans instance for k clusters
kmeans = KMeans(n_clusters=k, random_state=36)

# Create SilhouetteVisualizer instance with KMeans instance
# NOTE: this will automatically create a plot
visualizer = SilhouetteVisualizer(kmeans,
                                  colors='yellowbrick')

# Fit the visualizer
visualizer.fit(k_Xdata)

# label axes, etc
plt.title('Silhouette scores for k = {}'.format(k), size=18)
plt.xlabel('Silhouette Score', size=16)
plt.ylabel('Observation index', size=16)
plt.show()
```




[101]:
```
# look at the silhouette plot of this

###### set value of k
k = 2

# Create KMeans instance for k clusters
kmeans = KMeans(n_clusters=k, random_state=42)

# Create SilhouetteVisualizer instance with KMeans instance
# NOTE: this will automatically create a plot
visualizer = SilhouetteVisualizer(kmeans,
                                  colors='yellowbrick')

# Fit the visualizer
visualizer.fit(k_Xdata)

# label axes, etc
plt.title('Silhouette scores for k = {}'.format(k), size=18)
plt.xlabel('Silhouette Score', size=16)
plt.ylabel('Observation index', size=16)
plt.show()
```




[108]:
```
k_value = 2
```

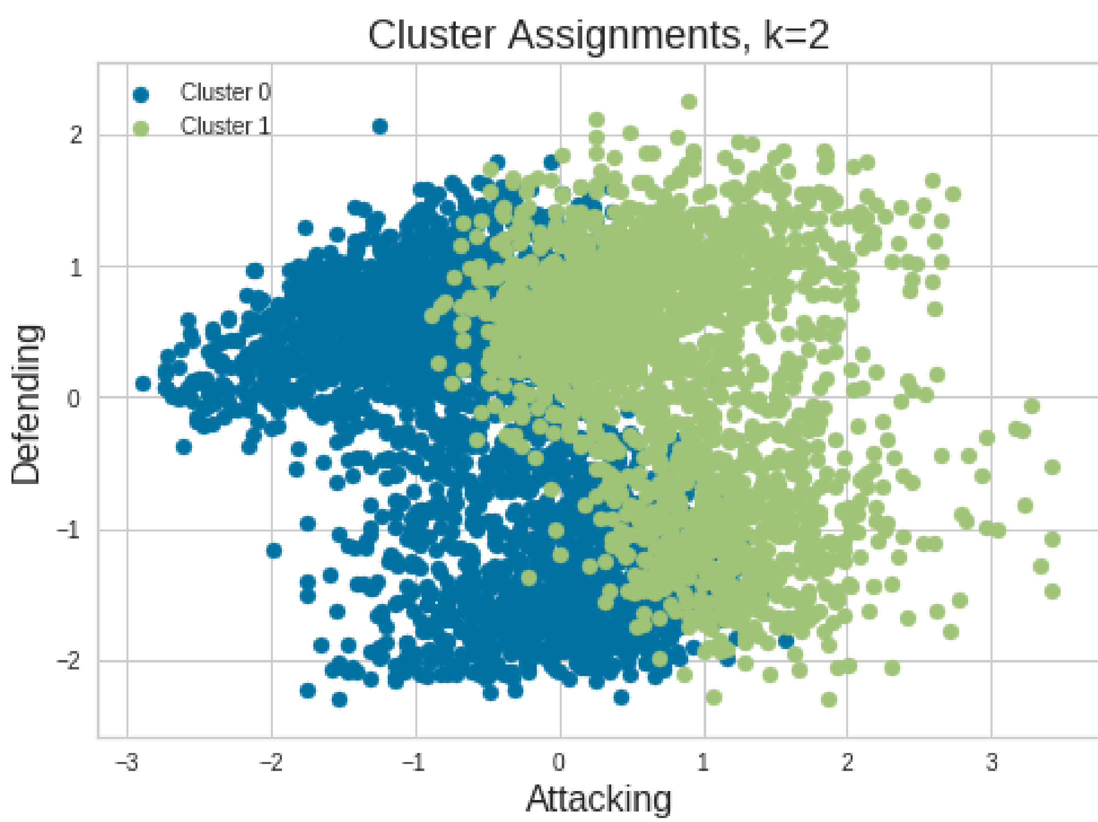
[109]:
```
# fit KMeans
KMeans_trained = KMeans(n_clusters= k_value).fit(k_Xdata)
# save labels to DF
k_Xdata['{} clusters'.format(k)] = KMeans_trained.labels_

k_Xdata.head()
```

[109]:
```
       shooting   passing  dribbling  defending  attacking     skill  \
291    1.373606  2.115543   1.465680   1.550464   1.830015  1.748668
501    1.937889  0.934444   1.876719  -0.849343   1.959068  1.377552
8871   1.020930 -0.246654  -0.795038  -0.736852   1.120221 -0.979033
12793  0.456648 -0.345079   0.129801  -1.580534   0.173830 -0.552250
7256  -1.377269 -0.541929  -1.206077   0.763027  -0.600490 -1.591375

       movement     power  mentality  goalkeeping  2 clusters
291    0.037498  1.944282   2.180614     0.948023           1
501    2.049414  1.870951   1.404908     0.148406           1
8871  -1.576714  0.990972   0.310965     0.281676           0
12793  1.090245  1.039860  -0.703419    -1.051018           0
```

```
7256  -0.430389  0.013218  -0.206172    0.814753          0
```

[110]:

```
for i in range(k_value):
    # subset to just observations in this cluster
    subset = k_Xdata[k_Xdata['{} clusters'.format(k)] == i]

    # plot this cluster
    # Each call to .scatter() will auto get a different color
    plt.scatter(subset['attacking'],
                subset['defending'],
                label='Cluster {}'.format(i))

# labels, etc
plt.xlabel('Attacking', size=16)
plt.ylabel('Defending', size=16)

plt.title('Cluster Assignments, k={}'.format(k), size=18)
plt.legend(loc='upper left', prop={'size': 10})

plt.show()
```



(k) Clustering this data enables us to understand the different categories of players we could get from this data. I would further like to analyse the different success rates of these players based on their mentality. That is one field that we didn't cover in the above analysis.

**——————-END OF HOMEWORK——————-**

```
[ ]:
```