January 15, 2024

[3]:
```
import pandas as pd
data = pd.read_csv("fifa22.csv")
data.head()
```

[3]:
```
                              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
```

1b) The unit of analysis in this data set is FIFA players. 1c) There are 19,630 observations and 20 features in the data set.

[4]:
```
data.shape
gender_list = data["gender"].tolist()
mcount = 0;
```

```
fcount = 0;

for i in gender_list:
    if i == 'M':
        mcount+=1
    else:
        fcount+=1

print(mcount)
print(fcount)
```

```
19239
391
```

1d) There are 19,239 males and 391 females. 1e) No, this is not representative of the real-world population of professional soccer players because all the playable characters in the video game are the top players that are popular, not every single player in the professional league is included in the game.

[5]:
```
#1f)
data= data.dropna(subset = ['passing'])
data.shape
```

[5]: (17450, 20)

[6]:
```
import statsmodels.api as sm
x = sm.add_constant(data[['passing', 'attacking', 'defending', 'skill']])
y = data['rank']
model = sm.OLS(y, x).fit()
print(model.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:                Wed, 29 Nov 2023   Prob (F-statistic):               0.00
Time:                        18:58:55   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]
------------------------------------------------------------------------------
const         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.
```

```
/opt/conda/envs/dsua-111/lib/python3.7/site-
packages/numpy/core/fromnumeric.py:2542: FutureWarning: Method .ptp is
deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

2b) 70.5% of the variation in rank is explained by our features. 2c) The features that are significant at the 1% level are attacking and defending. 2d) A 1-unit increase in "skill" is associated with a .0066 unit change in ranking. 3a) Skill is the feature that would not do well in predicting outofsample data because its p-value is too high to be statistically significant and the coefficient for skill is very small (0.0066), so it does not have a substantial impact on the dependent variable even if it was significant. The other features would do well.

[7]:
```python
x = data[['passing', 'attacking', 'defending', 'skill']]
x.head()
```

[7]:

	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

[8]:
```python
y = data['rank']
y.head()
```

[8]:
```
0    93
1    92
2    92
3    92
4    92
Name: rank, dtype: int64
```

[9]:
```python
from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.25,
 ↪random_state=123)
```

```
x_train.head()
```

[9]:

	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

[10]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
print(model.intercept_)
print(model.coef_)
```

```
25.167733064621757
[-0.02444506  0.61230756  0.17314968  0.00612364]
```

3e) The coeficient for trained attacking is 0.61230756 and for the full dataset it was 0.6109. Both regression models estimate similar coefficients

[11]:
```
#3f
pred = model.predict(x_valid)
print(pred[:3])
```

```
[64.57617047 72.78035994 70.46341746]
```

[12]:
```
from sklearn.metrics import mean_squared_error
import numpy as np
rmse = np.sqrt(mean_squared_error(y_valid, pred))
print(rmse)
```

```
3.744562639987198
```

This measures the error between the actual and predicted values, the lower the rmse the better becuase it means it is a better model beucase it means the predictions are closer to the actual.

The r-squared value showing that the model makes up for 70.5% of the variablity in the rank DV is relatively high which is a good sign of a good model. Also, considering how low the rsme is of around 3.74 it means the average magnitude of error in predicting rank is low. And the attacking and defending have 1% level of significance which means they have a good impact. Therefore, I think this model does a good job of predicting player rank?

[13]:
```
pref_foot = data["preferred_foot"]
print(pref_foot.value_counts())
```

```
Right    13044
Left      4406
Name: preferred_foot, dtype: int64
```

[14]:
```python
perc_right = (13044/(13044+4406)) *100
print(perc_right)
```

```
74.75071633237822
```

4b) 74.75% of players prefer their right foot.

[15]:
```python
x = data[ ["shooting", "passing", "dribbling", "defending", "attacking",␣
↪"skill", "movement", "power", "mentality", "goalkeeping"]]
x.head()
```

[15]:
```
   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
```

[16]:
```python
from sklearn.preprocessing import StandardScaler
scale=StandardScaler()
x_scaler = scale.fit_transform(x)
x_scaled = pd.DataFrame(x_scaler, columns = x.columns)
print(x_scaled.head(3))
```

```
   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

      power  mentality  goalkeeping
0  1.944282   2.180614     0.281676
1  2.066501   1.623697     1.481100
2  2.702042   1.822596     3.480141
```

[17]:
```python
y = data['preferred_foot']
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.30,␣
↪random_state=456)
x_train.head(3)
```

[17]:
```
       shooting  passing  dribbling  defending  attacking  skill  movement  \
17219      24.0     43.0       48.0  59.000000       37.2   38.8      58.2
```

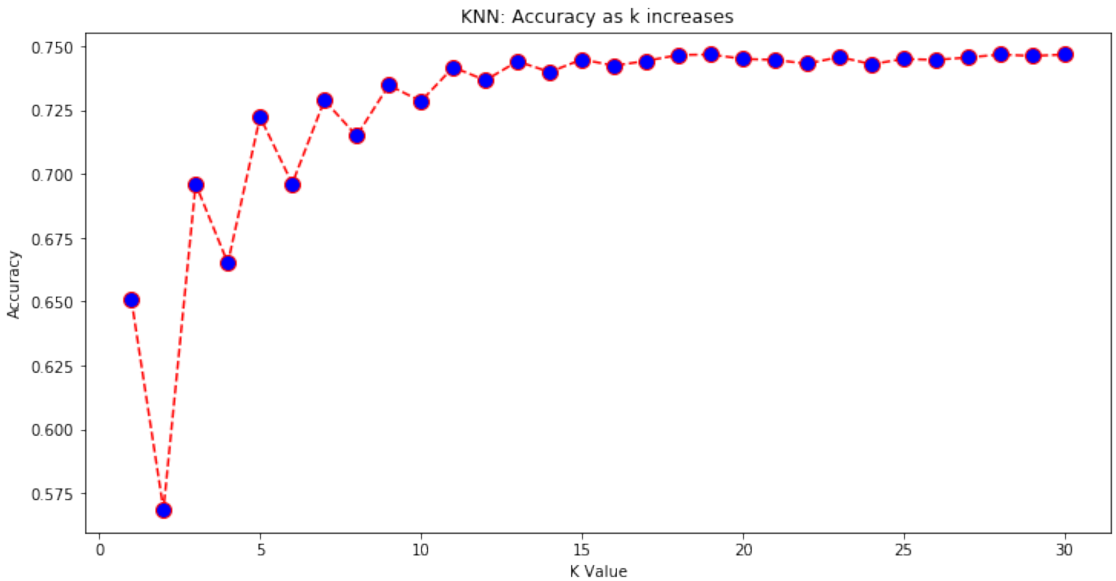
```
10931       46.0      61.0        70.0  58.666667        51.6    62.4       72.4
13667       57.0      59.0        64.0  35.333333        55.0    55.8       66.0

       power  mentality  goalkeeping
17219   53.6       42.5         11.2
10931   56.6       53.0         10.0
13667   58.6       52.5          7.6
```

[20]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

k_values = list(range(1, 31))
accuracy = []

for k in k_values:

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(x_train, y_train)

    y_pred = knn.predict(x_valid)

    accuracy.append(metrics.accuracy_score(y_valid, y_pred))
```

[21]:

```
#plt.plot(k_values, accuracies, marker='o')
plt.figure(figsize=(12, 6))
plt.plot(k_values, accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('KNN: Accuracy as k increases')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.show()
```



[22]:
```
classifier = KNeighborsClassifier(n_neighbors=15)
classifier.fit(x_train,y_train)
y_pred = classifier.predict(x_valid)
y_pred
```

[22]:
```
array(['Right', 'Right', 'Right', …, 'Right', 'Right', 'Right'],
      dtype=object)
```

[23]:
```
#4h
print(confusion_matrix(y_valid, y_pred))
print(classification_report(y_valid, y_pred))
```

```
[[ 120 1206]
 [ 129 3780]]
              precision    recall  f1-score   support

        Left       0.48      0.09      0.15      1326
       Right       0.76      0.97      0.85      3909

    accuracy                           0.74      5235
   macro avg       0.62      0.53      0.50      5235
weighted avg       0.69      0.74      0.67      5235
```

[24]:
```
cmat = confusion_matrix(y_valid, y_pred)

print('TP - True Positive: {}'.format(cmat[0,0]))
print('FP - False Positive: {}'.format(cmat[0,1]))
```

```
print('FN - False Negative: {}'.format(cmat[1,0]))
print('TN - True Negative: {}'.format(cmat[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.
 ↪sum(cmat))))
print('Misclassification Rate: {}'.format(np.divide(np.
 ↪sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))))
```

```
TP - True Positive: 120
FP - False Positive: 1206
FN - False Negative: 129
TN - True Negative: 3780
Accuracy Rate: 0.7449856733524355
Misclassification Rate: 0.25501432664756446
```

There are 1,071 players who actually prefer their left foot ("True Lefts") but were predicted to prefer their right foot.

4i) The recall for left is 0.09. Recall is the true positive rate and is the proportion of actual positive instances that were correctly identified by the model. The model is not doing good for identifying players who actually prefer their left foot. A lot of True Lefts are being incorrectly classified as having a right foot preference. 4j) The model does a bad job of predicting a player's preferred foot because it is bad at predicting true left foot preference the low recall value for left shows that and the low precision value shows that poor performance by the model.

[25]:
```
#5a
scalex = pd.DataFrame(x_scaled)
scalex.head()
```

[25]:
```
   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
```

[26]:
```
#5b
samp=scalex.sample(n=5000, random_state=2022)
samp.head()
```

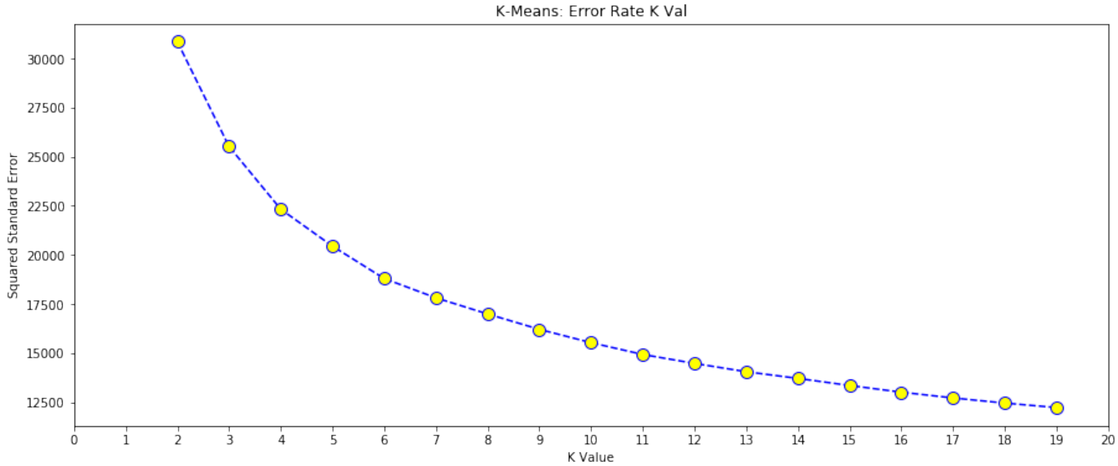
[26]:
```
        shooting   passing  dribbling  defending  attacking     skill  \
291     1.373606  2.115543   1.465680   1.550464   1.830015  1.748668
```

```
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
```

[34]:
```python
#5c
from sklearn.cluster import KMeans
error = list()
sil = list()

for k in range (2, 20):
    k_means = KMeans(n_clusters = k, random_state = 789)
    k_means.fit(samp)

    error.append(k_means.inertia_)
    score = metrics.silhouette_score(samp, k_means.labels_)
    sil.append(score)

print(error)
```

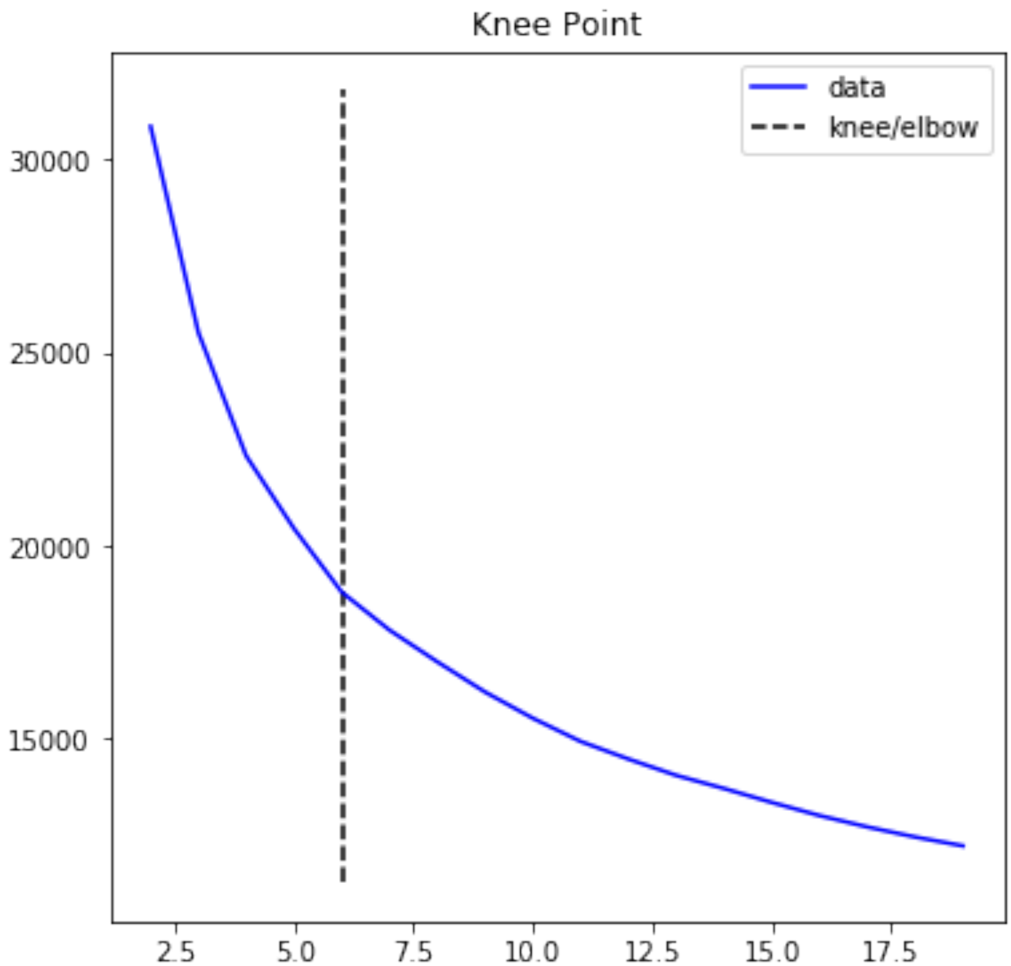
```
[30847.126857997708, 25514.51134236195, 22325.868999171504, 20446.193863352895,
18799.73556250355, 17818.103488273115, 16997.425516956362, 16215.351018257887,
15536.930308302319, 14936.765072641238, 14481.726819497208, 14059.160188748174,
13719.567782417698, 13359.277645318618, 13018.39026372852, 12724.739592119795,
12464.422720758232, 12236.010193739769]
```

[37]:
```python
#5d
plt.figure(figsize = (15, 6))
plt.plot(range(2, len(error) + 2), error, color = 'blue', linestyle =␣
 ↪'dashed',marker = 'o', markerfacecolor = 'yellow', markersize = 10)
plt.xticks(np.arange(0, 21, 1))
plt.title('K-Means: Error Rate K Val')
plt.xlabel('K Value')
plt.ylabel('Squared Standard Error')
plt.show()
```

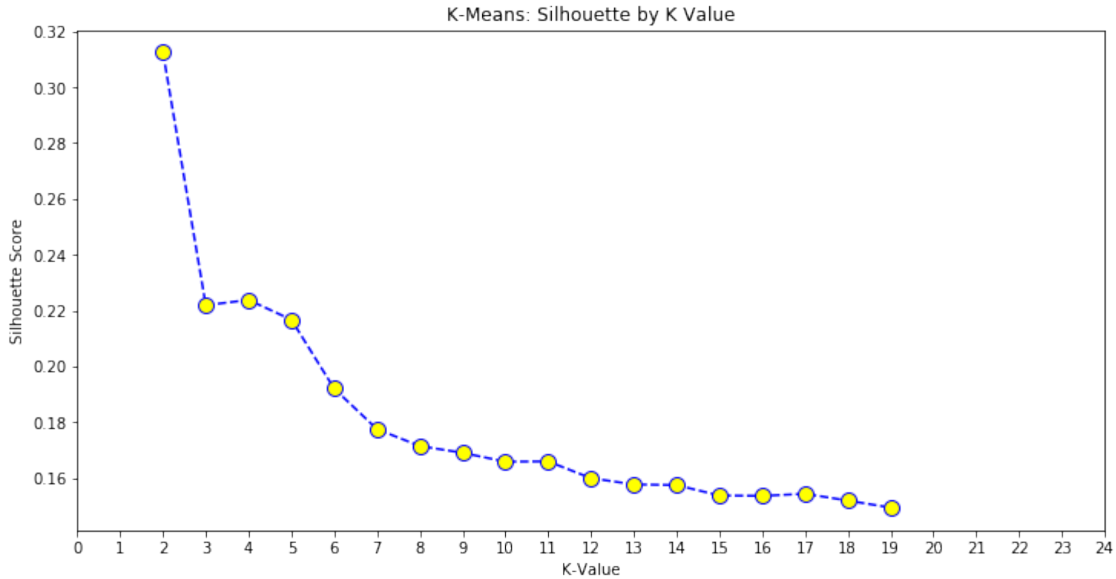


[39]:

```
#5e
import kneed
from kneed import KneeLocator

kneedle = KneeLocator(range(2, 20), error, curve = 'convex', direction='decreasing')
kneedle.elbow
kneedle.plot_knee()
```



[42]:
```
#5f
plt.figure(figsize = (12, 6))
plt.plot(range(2, len(sil) + 2), sil, color = 'blue', linestyle= 'dashed',␣
 ↪marker = 'o', markerfacecolor = 'yellow', markersize = 10)
plt.xticks(np.arange(0, 25, 1))
plt.title('K-Means: Silhouette by K Value')
plt.xlabel('K-Value')
plt.ylabel('Silhouette Score')
plt.show()
```



[48]:
```
#5g
k_means = KMeans(init = 'random',n_clusters = 5,n_init = 10,max_iter = 5000)
k_means.fit(samp)
clust_val = k_means.predict(scalex)
x.loc[:, 'cluster'] = clust_val.copy()
x.head()
```

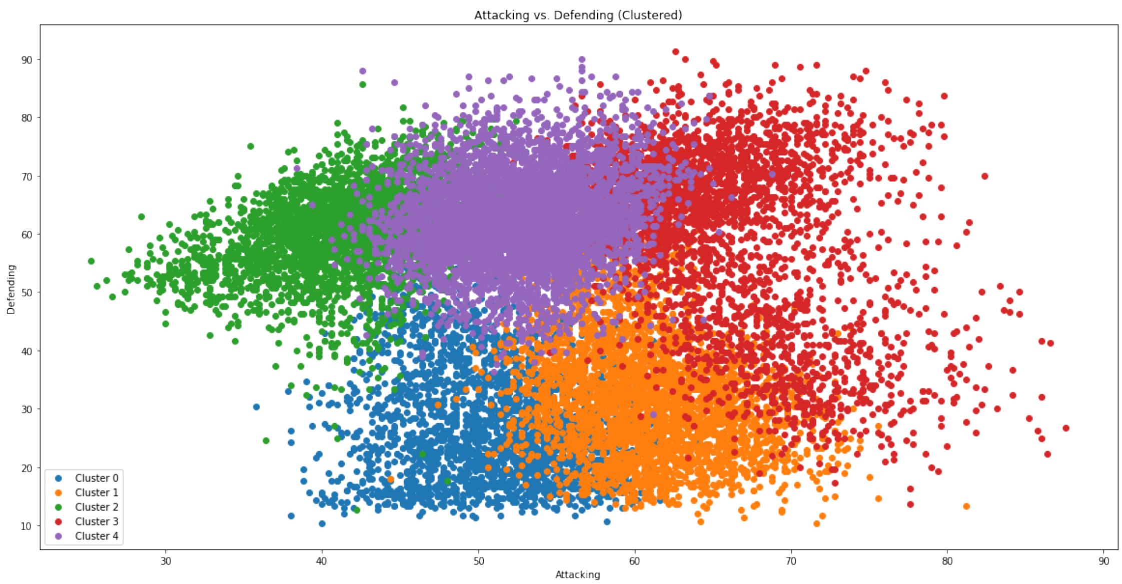
```
/opt/conda/envs/dsua-111/lib/python3.7/site-
packages/pandas/core/indexing.py:494: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

[48]:
```
   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  cluster
0  73.833333         10.8        3
1  69.166667         12.6        3
2  70.833333         15.6        3
3  73.500000         12.8        3
```

```
4  80.666667          10.2        3
```

[52]:

```
#5h
colors = {0: 'C0',1: 'C1',2: 'C2',3: 'C3',4: 'C4'}
plt.rc('figure', figsize = (20, 10))
for i in range (5):
    subset = x[x['cluster'] == i]
    plt.scatter(subset['attacking'],subset['defending'],label = 'Cluster {}'.
 ↪format(i), color = colors[i])

plt.title('Attacking vs. Defending (Clustered)')
plt.xlabel('Attacking')
plt.ylabel('Defending')
plt.legend(loc = 'lower left', prop = {'size': 10})
plt.show()
```



#5i A lot of clusters overlap which is not good, and this does not represent optimal clustering. To more accurately model the data, different k values should be used and more tests should be conducted. The clusters are too spread out. The red cluster is very spread out, and the purple is the most concentrated. The centroids are off. #5j If we run more regressions betwen different variables to see different associations that would be cool to help understand and divy up the data set and to focus in on certain factors. I would also be interested in running different clustering trying out different k-values.

[ ]: