Soccer Data Analysis

In [49]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
```

In [7]:
```
cnx = sqlite3.connect('C:\\Users\\Hadi\\database.sqlite')
df = pd.read_sql_query("SELECT * FROM Player_Attributes", cnx)
```

Exploring Data

In [8]:
```
df.columns
```

Out[8]:
```
Index(['id', 'player_fifa_api_id', 'player_api_id', 'date', 'overall_rating',
       'potential', 'preferred_foot', 'attacking_work_rate',
       'defensive_work_rate', 'crossing', 'finishing', 'heading_accuracy',
       'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy',
       'long_passing', 'ball_control', 'acceleration', 'sprint_speed',
       'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'stamina',
       'strength', 'long_shots', 'aggression', 'interceptions', 'positioning',
       'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle',
       'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning',
       'gk_reflexes'],
      dtype='object')
```

In [9]:
```
df.describe().transpose()
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
id	183978.0	91989.500000	53110.018250	1.0	45995.25	91989.5	137983.75	183978.0
player_fifa_api_id	183978.0	165671.524291	53851.094769	2.0	155798.00	183488.0	199848.00	234141.0
player_api_id	183978.0	135900.617324	136927.840510	2625.0	34763.00	77741.0	191080.00	750584.0
overall_rating	183142.0	68.600015	7.041139	33.0	64.00	69.0	73.00	94.0
potential	183142.0	73.460353	6.592271	39.0	69.00	74.0	78.00	97.0
crossing	183142.0	55.086883	17.242135	1.0	45.00	59.0	68.00	95.0
finishing	183142.0	49.921078	19.038705	1.0	34.00	53.0	65.00	97.0
heading_accuracy	183142.0	57.266023	16.488905	1.0	49.00	60.0	68.00	98.0
short_passing	183142.0	62.429672	14.194068	3.0	57.00	65.0	72.00	97.0
volleys	181265.0	49.468436	18.256618	1.0	35.00	52.0	64.00	93.0
dribbling	183142.0	59.175154	17.744688	1.0	52.00	64.0	72.00	97.0
curve	181265.0	52.965675	18.255788	2.0	41.00	56.0	67.00	94.0
free_kick_accuracy	183142.0	49.380950	17.831746	1.0	36.00	50.0	63.00	97.0
long_passing	183142.0	57.069880	14.394464	3.0	49.00	59.0	67.00	97.0
ball_control	183142.0	63.388879	15.196671	5.0	58.00	67.0	73.00	97.0
acceleration	183142.0	67.659357	12.983326	10.0	61.00	69.0	77.00	97.0
sprint_speed	183142.0	68.051244	12.569721	12.0	62.00	69.0	77.00	97.0
agility	181265.0	65.970910	12.954585	11.0	58.00	68.0	75.00	96.0
reactions	183142.0	66.103706	9.155408	17.0	61.00	67.0	72.00	96.0
balance	181265.0	65.189496	13.063188	12.0	58.00	67.0	74.00	96.0
shot_power	183142.0	61.808427	16.135143	2.0	54.00	65.0	73.00	97.0
jumping	181265.0	66.969045	11.006734	14.0	60.00	68.0	74.00	96.0
stamina	183142.0	67.038544	13.165262	10.0	61.00	69.0	76.00	96.0
strength	183142.0	67.424529	12.072280	10.0	60.00	69.0	76.00	96.0
long_shots	183142.0	53.339431	18.367025	1.0	41.00	58.0	67.00	96.0
aggression	183142.0	60.948046	16.089521	6.0	51.00	64.0	73.00	97.0
interceptions	183142.0	52.009271	19.450133	1.0	34.00	57.0	68.00	96.0
positioning	183142.0	55.786504	18.448292	2.0	45.00	60.0	69.00	96.0
vision	181265.0	57.873550	15.144086	1.0	49.00	60.0	69.00	97.0
penalties	183142.0	55.003986	15.546519	2.0	45.00	57.0	67.00	96.0
marking	183142.0	46.772242	21.227667	1.0	25.00	50.0	66.00	96.0
standing_tackle	183142.0	50.351257	21.483706	1.0	29.00	56.0	69.00	95.0
sliding_tackle	181265.0	48.001462	21.598778	2.0	25.00	53.0	67.00	95.0
gk_diving	183142.0	14.704393	16.865467	1.0	7.00	10.0	13.00	94.0
gk_handling	183142.0	16.063612	15.867382	1.0	8.00	11.0	15.00	93.0
gk_kicking	183142.0	20.998362	21.452980	1.0	8.00	12.0	15.00	97.0
gk_positioning	183142.0	16.132154	16.099175	1.0	8.00	11.0	15.00	96.0
gk_reflexes	183142.0	16.441439	17.198155	1.0	8.00	11.0	15.00	96.0

Data Cleaning: Handling Missing Data

In [10]:
```
df.isnull().any().any(), df.shape
```

Out[10]:
```
(True, (183978, 42))
```

In [11]:
```
df.isnull().sum(axis=0)
```

Out[11]:
```
id                      0
player_fifa_api_id      0
player_api_id           0
date                    0
overall_rating        836
potential             836
preferred_foot        836
attacking_work_rate  3230
defensive_work_rate   836
crossing              836
finishing             836
heading_accuracy      836
short_passing         836
volleys              2713
dribbling             836
curve                2713
free_kick_accuracy    836
long_passing          836
ball_control          836
acceleration          836
sprint_speed          836
agility              2713
reactions             836
balance              2713
shot_power            836
jumping              2713
stamina               836
strength              836
long_shots            836
aggression            836
interceptions         836
positioning           836
vision               2713
penalties             836
marking               836
standing_tackle       836
sliding_tackle       2713
gk_diving             836
gk_handling           836
gk_kicking            836
gk_positioning        836
gk_reflexes           836
dtype: int64
```

Fixing Null Values by Deleting Them

In [12]:
```
rows = df.shape[0]
df = df.dropna()
```

In [13]:
```
print(rows)
df.isnull().any().any(), df.shape
```

```
183978
```

Out[13]:
```
(False, (180354, 42))
```

In [14]:
```
#How many rows with NULL values?

rows - df.shape[0]
```

Out[14]:
```
3624
```

In [15]:
```
#Shuffle the rows of df so we get a distributed sample when we display top few rows

df = df.reindex(np.random.permutation(df.index))
```

Predicting: 'overall_rating' of a player

In [16]:
```
df.head(5)
```

Out[16]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attacking_work_rate	defens
74917	74918	211537	289145	2015-09-21 00:00:00	58.0	65.0	right	medium	
46997	46998	205670	363340	2013-09-20 00:00:00	69.0	76.0	right	medium	
124286	124287	220511	490184	2014-05-30 00:00:00	59.0	68.0	right	medium	
107876	107877	219740	481722	2015-03-27 00:00:00	64.0	73.0	right	medium	
157278	157279	211457	302012	2016-04-28 00:00:00	70.0	75.0	left	None	

5 rows × 42 columns

In [17]:
```
df[:10][['penalties', 'overall_rating']]
```

Out[17]:

	penalties	overall_rating
74917	23.0	58.0
46997	56.0	69.0
124286	62.0	59.0
107876	68.0	64.0
157278	68.0	70.0
60089	76.0	76.0
25910	49.0	71.0
107168	66.0	69.0
137647	52.0	71.0
125713	13.0	66.0

Feature Correlation Analysis

In [18]:
```
potentialFeatures = ['acceleration', 'curve', 'free_kick_accuracy', 'ball_control', 'shot_po
wer', 'stamina']
```

In [19]:
```
for f in potentialFeatures:
    related = df['overall_rating'].corr(df[f])
    print("%s: %f" % (f,related))
```

```
acceleration: 0.243998
curve: 0.357566
free_kick_accuracy: 0.349800
ball_control: 0.443991
shot_power: 0.428053
stamina: 0.325606
```

Data Visualization

In [20]:
```
cols = ['potential',  'crossing', 'finishing', 'heading_accuracy',
       'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy',
       'long_passing', 'ball_control', 'acceleration', 'sprint_speed',
       'agility', 'reactions', 'balance', 'shot_power', 'jumping', 'stamina',
       'strength', 'long_shots', 'aggression', 'interceptions', 'positioning',
       'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle',
       'gk_diving', 'gk_handling', 'gk_kicking', 'gk_positioning',
       'gk_reflexes']
```

In [21]:
```
# create a list containing Pearson's correlation between 'overall_rating' with each column i
n cols
correlations = [ df['overall_rating'].corr(df[f]) for f in cols ]
```

In [22]:
```
len(cols), len(correlations)
```

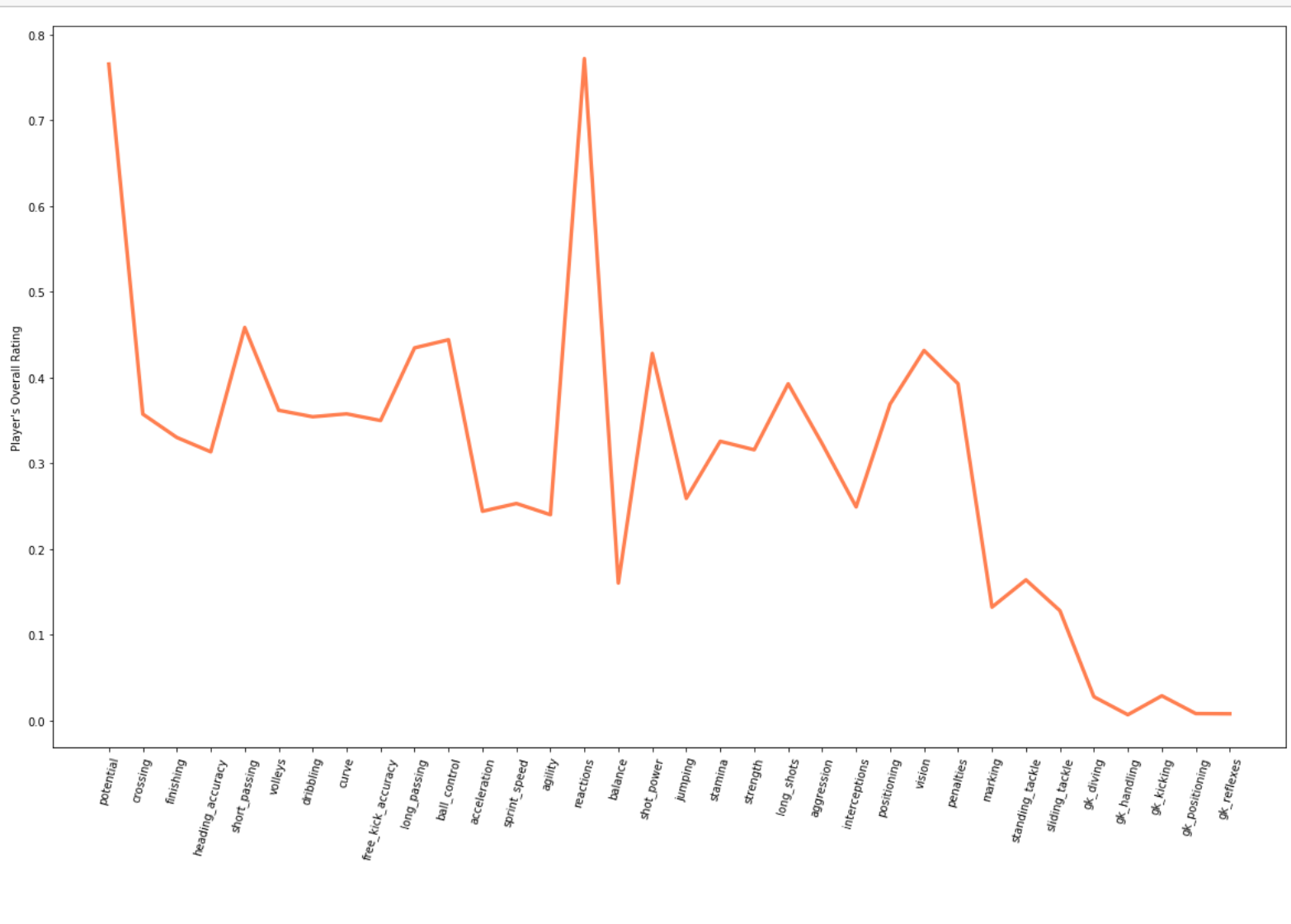
Out[22]:
```
(34, 34)
```

In [23]:
```
def plot_dataframe(df, y_label):
    color='coral'
    fig = plt.gcf()
    fig.set_size_inches(20, 12)
    plt.ylabel(y_label)

    ax = df.correlation.plot(linewidth=3.3, color=color)
    ax.set_xticks(df.index)
    ax.set_xticklabels(df.attributes, rotation=75); #Notice the ; (remove it and see what ha
ppens !)
    plt.show()
```

In [24]:
```
df2 = pd.DataFrame({'attributes': cols, 'correlation': correlations})
```

In [25]:
```
plot_dataframe(df2, 'Player\'s Overall Rating')
```



Clustering Players into Similar Groups

In [26]:
```
# Define the features you want to use for grouping players

select5features = ['gk_kicking', 'potential', 'marking', 'interceptions', 'standing_tackle']
select5features
```

Out[26]:
```
['gk_kicking', 'potential', 'marking', 'interceptions', 'standing_tackle']
```

In [34]:
```
df_select = df[select5features].copy(deep = True)
```

In [35]:
```
df_select.head()
```

Out[35]:

	gk_kicking	potential	marking	interceptions	standing_tackle
74917	56.0	65.0	15.0	19.0	14.0
46997	15.0	76.0	25.0	34.0	25.0
124286	7.0	68.0	22.0	45.0	36.0
107876	15.0	73.0	24.0	27.0	28.0
157278	7.0	75.0	15.0	21.0	20.0

Perform KMeans Clustering

In [36]:
```
data = scale(df_select)

# Define number of clusters
noOfClusters = 4

# Train a model
model = KMeans(init='k-means++', n_clusters=noOfClusters, n_init=20).fit(data)
```

In [37]:
```
print(90*'_')
print("\nCount of players in each cluster")
print(90*'_')

pd.value_counts(model.labels_, sort=False)
```

```
__________________________________________________________________________________________

Count of players in each cluster
__________________________________________________________________________________________
```

Out[37]:
```
0    55904
1    50254
2    50409
3    23787
dtype: int64
```

In [46]:
```
def pd_centers(featuresUsed, centers):
        from itertools import cycle, islice
        from pandas.tools.plotting import parallel_coordinates
        import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np

        colNames = list(featuresUsed)
        colNames.append('prediction')

        # Zip with a column called 'prediction' (index)
        Z = [np.append(A, index) for index, A in enumerate(centers)]

        # Convert to pandas for plotting
        P = pd.DataFrame(Z, columns=colNames)
        P['prediction'] = P['prediction'].astype(int)
        return P

def parallel_plot(data):
        from itertools import cycle, islice
        from pandas.tools.plotting import parallel_coordinates
        import matplotlib.pyplot as plt

        my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(data)))
        plt.figure(figsize=(15,8)).gca().axes.set_ylim([-2.5,+2.5])
        parallel_coordinates(data, 'prediction', color = my_colors, marker='o')
```

In [47]:
```
P = pd_centers(featuresUsed=select5features, centers=model.cluster_centers_)
P
```

Out[47]:

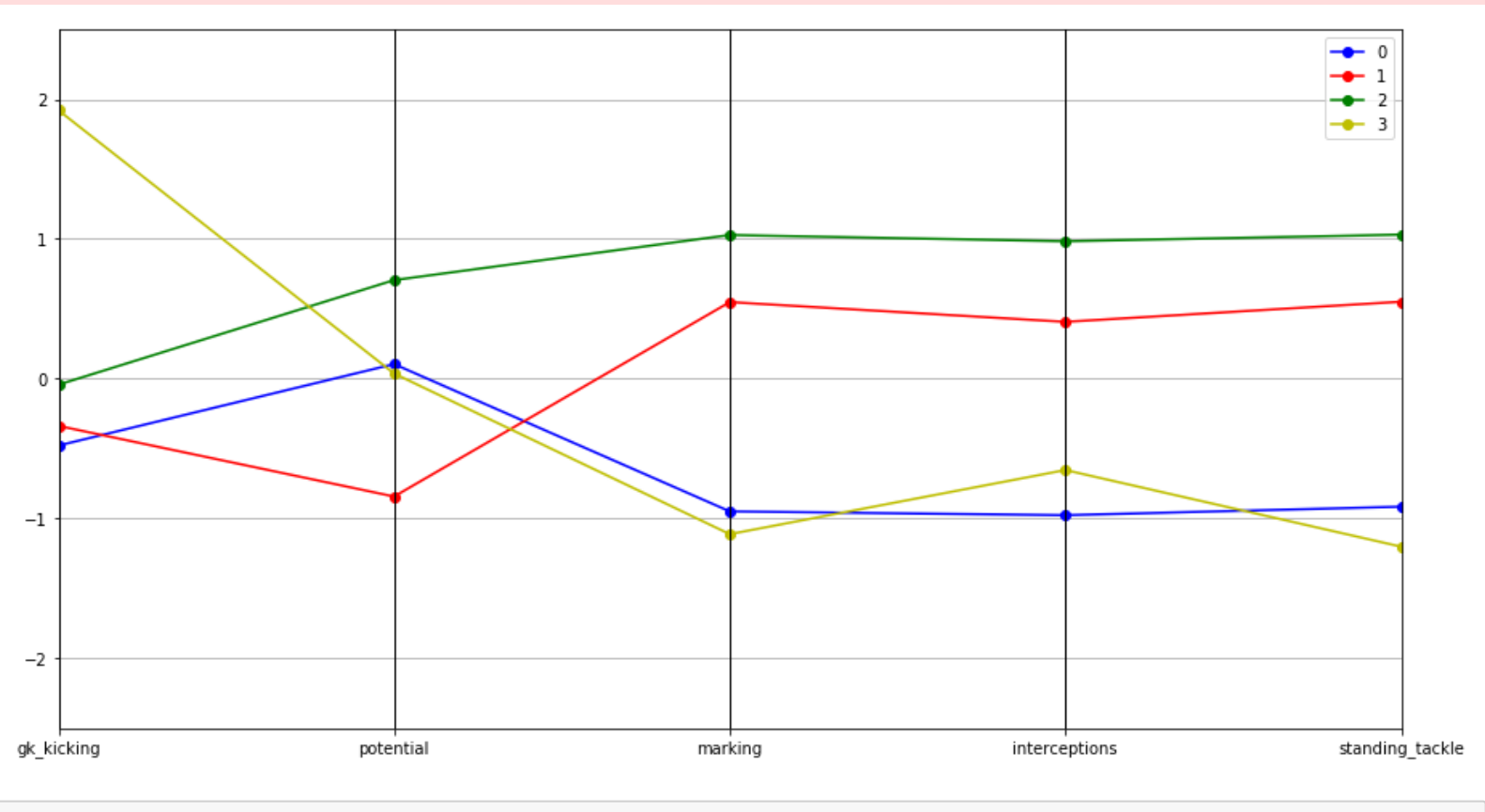
	gk_kicking	potential	marking	interceptions	standing_tackle	prediction
0	-0.477139	0.105625	-0.947740	-0.975241	-0.914225	0
1	-0.336207	-0.841539	0.548815	0.407592	0.551337	1
2	-0.041782	0.706043	1.028771	0.983736	1.031231	2
3	1.920531	0.038315	-1.110692	-0.652083	-1.199968	3

Visualization of Clusters

In [50]:
```
# For plotting the graph inside the notebook itself, we use the following command

%matplotlib inline
parallel_plot(P)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:26: FutureWarning: 'pandas.t
ools.plotting.parallel_coordinates' is deprecated, import 'pandas.plotting.parallel_coordinat
es' instead.
```



In [ ]: