BA-S19-Project

April 26, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: dataset = pd.read_csv("data.csv")
```

Now I would like to clean up the data. Several steps should be taken[U+FF1A]

- seperate the dataset into parts to extract metrics to evaluate a certain player
- parse the data into appropriate datatypes
- eliminate all NaN and NAs in the dataset

```
In [3]: dataset.head()
```

```
Out[3]:    Unnamed: 0      ID               Name  Age  \
        0           0  158023           L. Messi   31
        1           1   20801  Cristiano Ronaldo   33
        2           2  190871          Neymar Jr   26
        3           3  193080             De Gea   27
        4           4  192985       K. De Bruyne   27

                                                   Photo Nationality  \
        0  https://cdn.sofifa.org/players/4/19/158023.png   Argentina
        1   https://cdn.sofifa.org/players/4/19/20801.png    Portugal
        2  https://cdn.sofifa.org/players/4/19/190871.png      Brazil
        3  https://cdn.sofifa.org/players/4/19/193080.png       Spain
        4  https://cdn.sofifa.org/players/4/19/192985.png     Belgium

                                          Flag  Overall  Potential  \
        0  https://cdn.sofifa.org/flags/52.png       94         94
        1  https://cdn.sofifa.org/flags/38.png       94         94
        2  https://cdn.sofifa.org/flags/54.png       92         93
        3  https://cdn.sofifa.org/flags/45.png       91         93
        4   https://cdn.sofifa.org/flags/7.png       91         92
```

```
                   Club      ...       Composure Marking StandingTackle  \
0          FC Barcelona      ...            96.0    33.0           28.0
1              Juventus      ...            95.0    28.0           31.0
2   Paris Saint-Germain      ...            94.0    27.0           24.0
3     Manchester United      ...            68.0    15.0           21.0
4       Manchester City      ...            88.0    68.0           58.0

   SlidingTackle GKDiving  GKHandling  GKKicking  GKPositioning GKReflexes  \
0           26.0      6.0        11.0       15.0           14.0        8.0
1           23.0      7.0        11.0       15.0           14.0       11.0
2           33.0      9.0         9.0       15.0           15.0       11.0
3           13.0     90.0        85.0       87.0           88.0       94.0
4           51.0     15.0        13.0        5.0           10.0       13.0

  Release Clause
0        €226.5M
1        €127.1M
2        €228.1M
3        €138.6M
4        €196.4M

[5 rows x 89 columns]
```

In [4]: `dataset.shape`

Out[4]: `(18207, 89)`

In [5]: `dataset.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 89 columns):
Unnamed: 0                  18207 non-null int64
ID                          18207 non-null int64
Name                        18207 non-null object
Age                         18207 non-null int64
Photo                       18207 non-null object
Nationality                 18207 non-null object
Flag                        18207 non-null object
Overall                     18207 non-null int64
Potential                   18207 non-null int64
Club                        17966 non-null object
Club Logo                   18207 non-null object
Value                       18207 non-null object
Wage                        18207 non-null object
Special                     18207 non-null int64
Preferred Foot              18159 non-null object
International Reputation    18159 non-null float64
Weak Foot                   18159 non-null float64
```

```
Skill Moves                 18159 non-null float64
Work Rate                   18159 non-null object
Body Type                   18159 non-null object
Real Face                   18159 non-null object
Position                    18147 non-null object
Jersey Number               18147 non-null float64
Joined                      16654 non-null object
Loaned From                 1264 non-null object
Contract Valid Until        17918 non-null object
Height                      18159 non-null object
Weight                      18159 non-null object
LS                          16122 non-null object
ST                          16122 non-null object
RS                          16122 non-null object
LW                          16122 non-null object
LF                          16122 non-null object
CF                          16122 non-null object
RF                          16122 non-null object
RW                          16122 non-null object
LAM                         16122 non-null object
CAM                         16122 non-null object
RAM                         16122 non-null object
LM                          16122 non-null object
LCM                         16122 non-null object
CM                          16122 non-null object
RCM                         16122 non-null object
RM                          16122 non-null object
LWB                         16122 non-null object
LDM                         16122 non-null object
CDM                         16122 non-null object
RDM                         16122 non-null object
RWB                         16122 non-null object
LB                          16122 non-null object
LCB                         16122 non-null object
CB                          16122 non-null object
RCB                         16122 non-null object
RB                          16122 non-null object
Crossing                    18159 non-null float64
Finishing                   18159 non-null float64
HeadingAccuracy             18159 non-null float64
ShortPassing                18159 non-null float64
Volleys                     18159 non-null float64
Dribbling                   18159 non-null float64
Curve                       18159 non-null float64
FKAccuracy                  18159 non-null float64
LongPassing                 18159 non-null float64
BallControl                 18159 non-null float64
Acceleration                18159 non-null float64
```

```
SprintSpeed                 18159 non-null float64
Agility                     18159 non-null float64
Reactions                   18159 non-null float64
Balance                     18159 non-null float64
ShotPower                   18159 non-null float64
Jumping                     18159 non-null float64
Stamina                     18159 non-null float64
Strength                    18159 non-null float64
LongShots                   18159 non-null float64
Aggression                  18159 non-null float64
Interceptions               18159 non-null float64
Positioning                 18159 non-null float64
Vision                      18159 non-null float64
Penalties                   18159 non-null float64
Composure                   18159 non-null float64
Marking                     18159 non-null float64
StandingTackle              18159 non-null float64
SlidingTackle               18159 non-null float64
GKDiving                    18159 non-null float64
GKHandling                  18159 non-null float64
GKKicking                   18159 non-null float64
GKPositioning               18159 non-null float64
GKReflexes                  18159 non-null float64
Release Clause              16643 non-null object
dtypes: float64(38), int64(6), object(45)
memory usage: 12.4+ MB
```

Many of these data are not what we want: we want to keep the player information and their ablity metrics, and put these in different dataframes.

According to the information shown above, a solution is to first eliminate the data columns with abnormal numbers of entries, and eliminate NAs afterwards.

The first step is to eliminate the scores that identify what the player would be like if he had changed to different positions.

```
In [6]: #Drop one column from the dataset
        #@param: df : dataframe to edit
        #         String: the label
        #@return: the edited dataframe
        def dataDrop(df,String):
            x = df[df.columns.drop(String)]
            return x
```

```
In [7]: #preprocess the data
        #change this method if want to change parameters
        #@param: df : dataframe to edit
        #@return: the edited dataframe
        def eliminateColumns(df,lst):
```

```
    for i in lst:
        df = dataDrop(df,i)
    return df
```

```
In [8]: eliminate = ['LS','ST','RS','LW','LF','CF','RF','RW','LAM','CAM','RAM','LM','LCM','CM','
                     'CB','RCB','RB','Special','International Reputation','Work Rate','Body Type
                     'Contract Valid Until','Release Clause','Wage','Flag','Nationality','Club L
        dataset_new = eliminateColumns(dataset,eliminate).dropna()
```

```
In [9]: ## Clean up the weight column
        dataset_new['WeightKG'] = dataset_new['Weight'].str.rstrip(to_strip='lbs').astype('float
        dataset_new = dataDrop(dataset_new,'Weight')
```

```
In [10]: ## Clean up the Height column
         temp = dataset_new['Height'].str.split('\'',expand=True).astype('float64')
         dataset_new['HeightCM'] = temp[0] * 30.48 + temp[1] * 2.54
         dataset_new = dataDrop(dataset_new,'Height')
```

```
In [11]: ## Clean up the value
         temp = dataset_new['Value'].str.lstrip(to_strip='€')
         temp_index =  dataset_new['Unnamed: 0']
         ## Use RegEx to convert to numerical values
         import re
         #Check if the string ends with M or K
         for i in temp_index:
             x = re.search("M$", temp[i])
             y = re.search("K$",temp[i])
             if (x):
                 temp[i] = float(temp[i].rstrip('M')) * 1000000
             elif (y):
                 temp[i] = float(temp[i].rstrip('K')) * 1000
             else:
                 temp[i] = float(temp[i])

         dataset_new['Value'] = temp
```

```
In [12]: ## Eliminate outliers
         outlier = np.where(dataset_new['Value'] <= 1000)
         dataset_new.drop(dataset_new.index[outlier],inplace = True)
```

```
In [13]: ## Seperate the dataset
         dataset_GK = dataset_new.copy()
         dataset_General = dataset_new.copy()
         temp = np.where(dataset_GK['Position'] != 'GK')
         dataset_GK.drop(dataset_GK.index[temp],inplace = True)
         temp = np.where(dataset_General['Position'] == 'GK')
         dataset_General.drop(dataset_General.index[temp],inplace = True)
```

```
In [14]: dataset_GK.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1989 entries, 3 to 18198
Data columns (total 50 columns):
Unnamed: 0         1989 non-null int64
ID                 1989 non-null int64
Name               1989 non-null object
Age                1989 non-null int64
Photo              1989 non-null object
Overall            1989 non-null int64
Potential          1989 non-null int64
Club               1989 non-null object
Value              1989 non-null object
Preferred Foot     1989 non-null object
Weak Foot          1989 non-null float64
Skill Moves        1989 non-null float64
Position           1989 non-null object
Jersey Number      1989 non-null float64
Crossing           1989 non-null float64
Finishing          1989 non-null float64
HeadingAccuracy    1989 non-null float64
ShortPassing       1989 non-null float64
Volleys            1989 non-null float64
Dribbling          1989 non-null float64
Curve              1989 non-null float64
FKAccuracy         1989 non-null float64
LongPassing        1989 non-null float64
BallControl        1989 non-null float64
Acceleration       1989 non-null float64
SprintSpeed        1989 non-null float64
Agility            1989 non-null float64
Reactions          1989 non-null float64
Balance            1989 non-null float64
ShotPower          1989 non-null float64
Jumping            1989 non-null float64
Stamina            1989 non-null float64
Strength           1989 non-null float64
LongShots          1989 non-null float64
Aggression         1989 non-null float64
Interceptions      1989 non-null float64
Positioning        1989 non-null float64
Vision             1989 non-null float64
Penalties          1989 non-null float64
Composure          1989 non-null float64
Marking            1989 non-null float64
StandingTackle     1989 non-null float64
SlidingTackle      1989 non-null float64
GKDiving           1989 non-null float64
GKHandling         1989 non-null float64
```

```
GKKicking          1989 non-null float64
GKPositioning      1989 non-null float64
GKReflexes         1989 non-null float64
WeightKG           1989 non-null float64
HeightCM           1989 non-null float64
dtypes: float64(39), int64(5), object(6)
memory usage: 792.5+ KB
```

```
In [15]: dataset_General.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15918 entries, 0 to 18206
Data columns (total 50 columns):
Unnamed: 0         15918 non-null int64
ID                 15918 non-null int64
Name               15918 non-null object
Age                15918 non-null int64
Photo              15918 non-null object
Overall            15918 non-null int64
Potential          15918 non-null int64
Club               15918 non-null object
Value              15918 non-null object
Preferred Foot     15918 non-null object
Weak Foot          15918 non-null float64
Skill Moves        15918 non-null float64
Position           15918 non-null object
Jersey Number      15918 non-null float64
Crossing           15918 non-null float64
Finishing          15918 non-null float64
HeadingAccuracy    15918 non-null float64
ShortPassing       15918 non-null float64
Volleys            15918 non-null float64
Dribbling          15918 non-null float64
Curve              15918 non-null float64
FKAccuracy         15918 non-null float64
LongPassing        15918 non-null float64
BallControl        15918 non-null float64
Acceleration       15918 non-null float64
SprintSpeed        15918 non-null float64
Agility            15918 non-null float64
Reactions          15918 non-null float64
Balance            15918 non-null float64
ShotPower          15918 non-null float64
Jumping            15918 non-null float64
Stamina            15918 non-null float64
Strength           15918 non-null float64
LongShots          15918 non-null float64
```

```
Aggression         15918 non-null float64
Interceptions      15918 non-null float64
Positioning        15918 non-null float64
Vision             15918 non-null float64
Penalties          15918 non-null float64
Composure          15918 non-null float64
Marking            15918 non-null float64
StandingTackle     15918 non-null float64
SlidingTackle      15918 non-null float64
GKDiving           15918 non-null float64
GKHandling         15918 non-null float64
GKKicking          15918 non-null float64
GKPositioning      15918 non-null float64
GKReflexes         15918 non-null float64
WeightKG           15918 non-null float64
HeightCM           15918 non-null float64
dtypes: float64(39), int64(5), object(6)
memory usage: 6.2+ MB
```

In [16]: `dataset_General.columns`

```
Out[16]: Index(['Unnamed: 0', 'ID', 'Name', 'Age', 'Photo', 'Overall', 'Potential',
                'Club', 'Value', 'Preferred Foot', 'Weak Foot', 'Skill Moves',
                'Position', 'Jersey Number', 'Crossing', 'Finishing', 'HeadingAccuracy',
                'ShortPassing', 'Volleys', 'Dribbling', 'Curve', 'FKAccuracy',
                'LongPassing', 'BallControl', 'Acceleration', 'SprintSpeed', 'Agility',
                'Reactions', 'Balance', 'ShotPower', 'Jumping', 'Stamina', 'Strength',
                'LongShots', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
                'Penalties', 'Composure', 'Marking', 'StandingTackle', 'SlidingTackle',
                'GKDiving', 'GKHandling', 'GKKicking', 'GKPositioning', 'GKReflexes',
                'WeightKG', 'HeightCM'],
               dtype='object')
```

In [17]: `dataset_new.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17907 entries, 0 to 18206
Data columns (total 50 columns):
Unnamed: 0         17907 non-null int64
ID                 17907 non-null int64
Name               17907 non-null object
Age                17907 non-null int64
Photo              17907 non-null object
Overall            17907 non-null int64
Potential          17907 non-null int64
Club               17907 non-null object
Value              17907 non-null object
Preferred Foot     17907 non-null object
```

```
Weak Foot          17907 non-null float64
Skill Moves        17907 non-null float64
Position           17907 non-null object
Jersey Number      17907 non-null float64
Crossing           17907 non-null float64
Finishing          17907 non-null float64
HeadingAccuracy    17907 non-null float64
ShortPassing       17907 non-null float64
Volleys            17907 non-null float64
Dribbling          17907 non-null float64
Curve              17907 non-null float64
FKAccuracy         17907 non-null float64
LongPassing        17907 non-null float64
BallControl        17907 non-null float64
Acceleration       17907 non-null float64
SprintSpeed        17907 non-null float64
Agility            17907 non-null float64
Reactions          17907 non-null float64
Balance            17907 non-null float64
ShotPower          17907 non-null float64
Jumping            17907 non-null float64
Stamina            17907 non-null float64
Strength           17907 non-null float64
LongShots          17907 non-null float64
Aggression         17907 non-null float64
Interceptions      17907 non-null float64
Positioning        17907 non-null float64
Vision             17907 non-null float64
Penalties          17907 non-null float64
Composure          17907 non-null float64
Marking            17907 non-null float64
StandingTackle     17907 non-null float64
SlidingTackle      17907 non-null float64
GKDiving           17907 non-null float64
GKHandling         17907 non-null float64
GKKicking          17907 non-null float64
GKPositioning      17907 non-null float64
GKReflexes         17907 non-null float64
WeightKG           17907 non-null float64
HeightCM           17907 non-null float64
dtypes: float64(39), int64(5), object(6)
memory usage: 7.0+ MB
```

This is a truncated dataset with 17907 observations.

Then we need to further clean-up the dataset: mainly by extracting the numerical values, and visualize the performance of a player.

To put the performances into less dimensions, we use the following methods:

Because they are already on a scale from 1 to 100, there is no need to standardaize them.

In [17]:
```
def attacking(data):
    return int(data[['Aggression','Finishing', 'Volleys', 'FKAccuracy',
                              'ShotPower','LongShots','Penalties',
                           'HeadingAccuracy','Curve']].mean())

def defending(data):
    return int(data[['Marking', 'StandingTackle',
                              'SlidingTackle','Interceptions']].mean())

def ballControl(data):
    return int(data[['Dribbling','BallControl']].mean())

def mentality(data):
    return int(data[['Positioning','Vision','Composure']].mean())

def passing(data):
    return int(data[['Crossing', 'ShortPassing','LongPassing']].mean())

def mobility(data):
    return int(data[['Acceleration', 'SprintSpeed',
                              'Agility','Reactions']].mean())
def physical(data):
    return int(data[['Balance', 'Jumping', 'Stamina','Strength',]].mean())




# adding these categories to the data
General_Polar = dataset_General.copy()
General_Polar['Defending'] = dataset_General.apply(defending, axis = 1)
General_Polar['Control'] = dataset_General.apply(ballControl, axis = 1)
General_Polar['Mentality'] = dataset_General.apply(mentality, axis = 1)
General_Polar['Passing'] = dataset_General.apply(passing, axis = 1)
General_Polar['Mobility'] = dataset_General.apply(mobility, axis = 1)
General_Polar['Physical'] = dataset_General.apply(physical, axis = 1)
General_Polar['Attacking'] = dataset_General.apply(attacking, axis = 1)
General_players = General_Polar[['Name','Defending','Control','Mentality','Passing',
                'Mobility','Physical','Overall','Attacking','Age','Club']]

General_players.head()
```

Out[17]:
```
                 Name  Defending  Control  Mentality  Passing  Mobility  \
0            L. Messi         27       96         94       87        90
1   Cristiano Ronaldo         27       91         90       80        90
2           Neymar Jr         30       95         90       80        93
4        K. De Bruyne         59       88         89       92        81
```

```
5           E. Hazard        31       94          89       84         91

   Physical  Overall  Attacking  Age                 Club
0        73       94         82   31         FC Barcelona
1        83       94         84   33             Juventus
2        68       92         78   26  Paris Saint-Germain
4        76       91         80   27      Manchester City
5        74       91         76   27              Chelsea
```

```
In [18]: General_players.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15918 entries, 0 to 18206
Data columns (total 11 columns):
Name          15918 non-null object
Defending     15918 non-null int64
Control       15918 non-null int64
Mentality     15918 non-null int64
Passing       15918 non-null int64
Mobility      15918 non-null int64
Physical      15918 non-null int64
Overall       15918 non-null int64
Attacking     15918 non-null int64
Age           15918 non-null int64
Club          15918 non-null object
dtypes: int64(9), object(2)
memory usage: 1.5+ MB
```

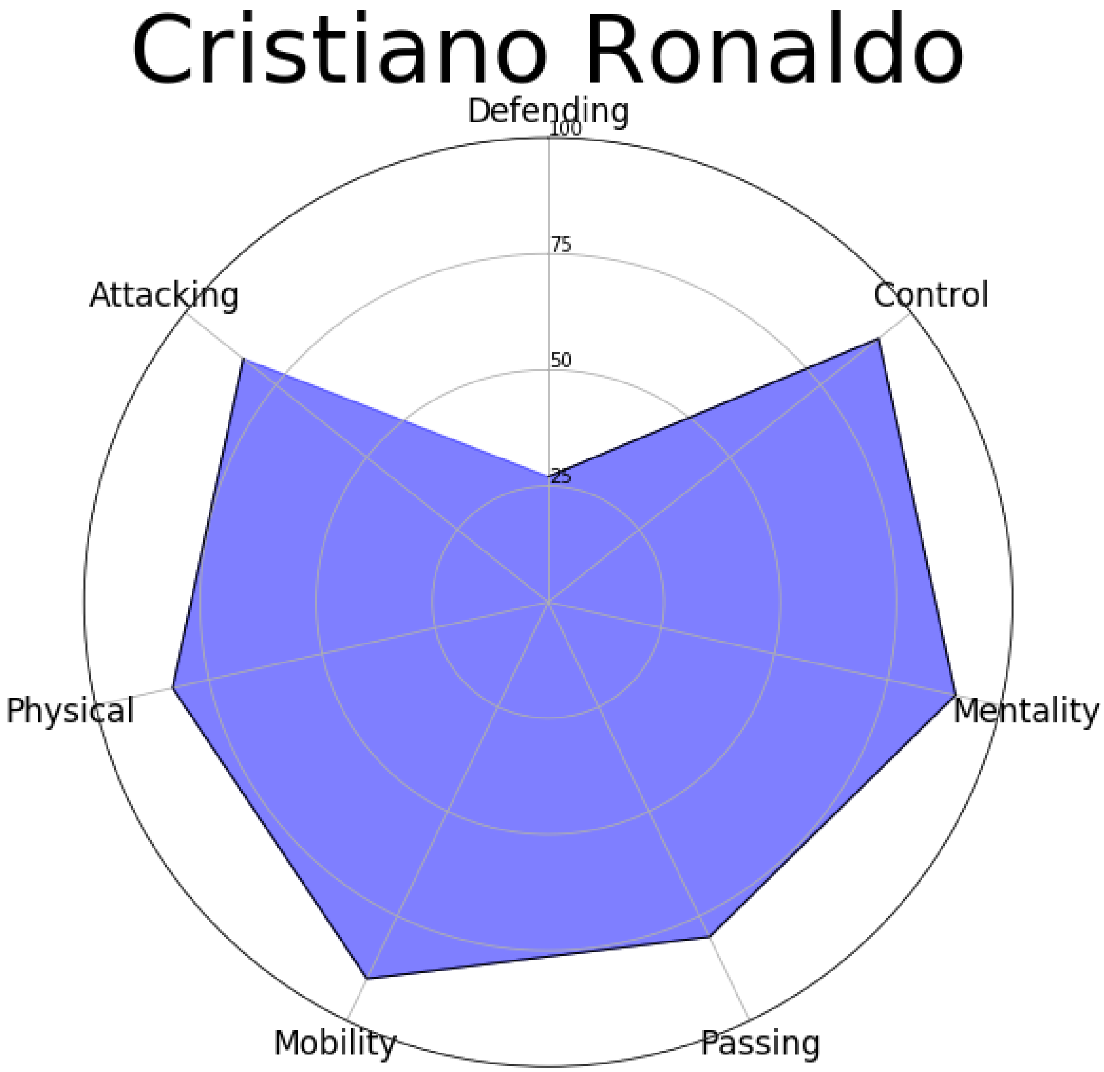
```
In [19]: General_players_graph = dataDrop(General_players,'Age')
         General_players_graph = dataDrop(General_players_graph,'Overall')
         General_players_graph = dataDrop(General_players_graph,'Club')
         General_players_graph.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15918 entries, 0 to 18206
Data columns (total 8 columns):
Name          15918 non-null object
Defending     15918 non-null int64
Control       15918 non-null int64
Mentality     15918 non-null int64
Passing       15918 non-null int64
Mobility      15918 non-null int64
Physical      15918 non-null int64
Attacking     15918 non-null int64
dtypes: int64(7), object(1)
memory usage: 1.1+ MB
```

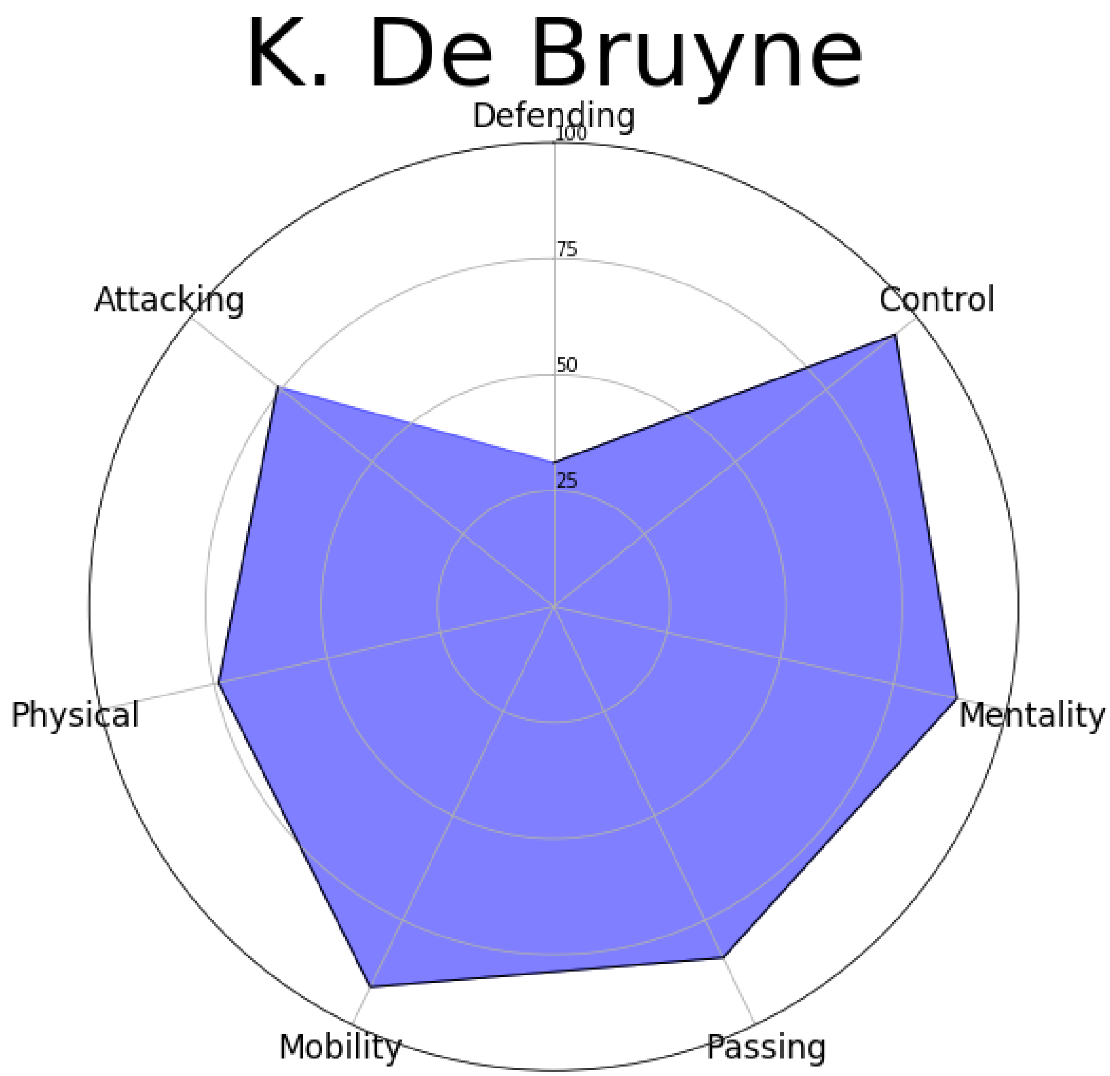
In [26]:
```
from math import pi
# defining a polar graph
def graphPolarGeneral(id):
    plt.figure(figsize=(16,9))
    categories=list(General_players_graph)[1:]
    N = len(categories)
    angles = [n / float(N) * 2 * pi for n in range(N)]
    ax = plt.subplot(111, projection='polar')
    ax.set_theta_offset(pi / 2)
    ax.set_theta_direction(-1)
    plt.xticks(angles, categories, color= '#000000', size=17)
    ax.set_rlabel_position(0)
    plt.yticks([25,50,75,100], ["25","50","75","100"], color= '#000000', size= 10)
    plt.ylim(0,100)
    values =  General_players_graph.loc[General_players_graph.index[id]].drop('Name').v
    ax.plot(angles, values, color= '#000000', linewidth=1, linestyle='solid')
    ax.fill(angles, values, color= '#0000ff', alpha=0.5)
    axes_coords = [0, 0, 1, 1]
    plt.title(General_players['Name'][id], size=50, color= '#000000')
    plt.savefig(str(id)+'.png')
```

In [27]:
```
graphPolarGeneral(0)
```

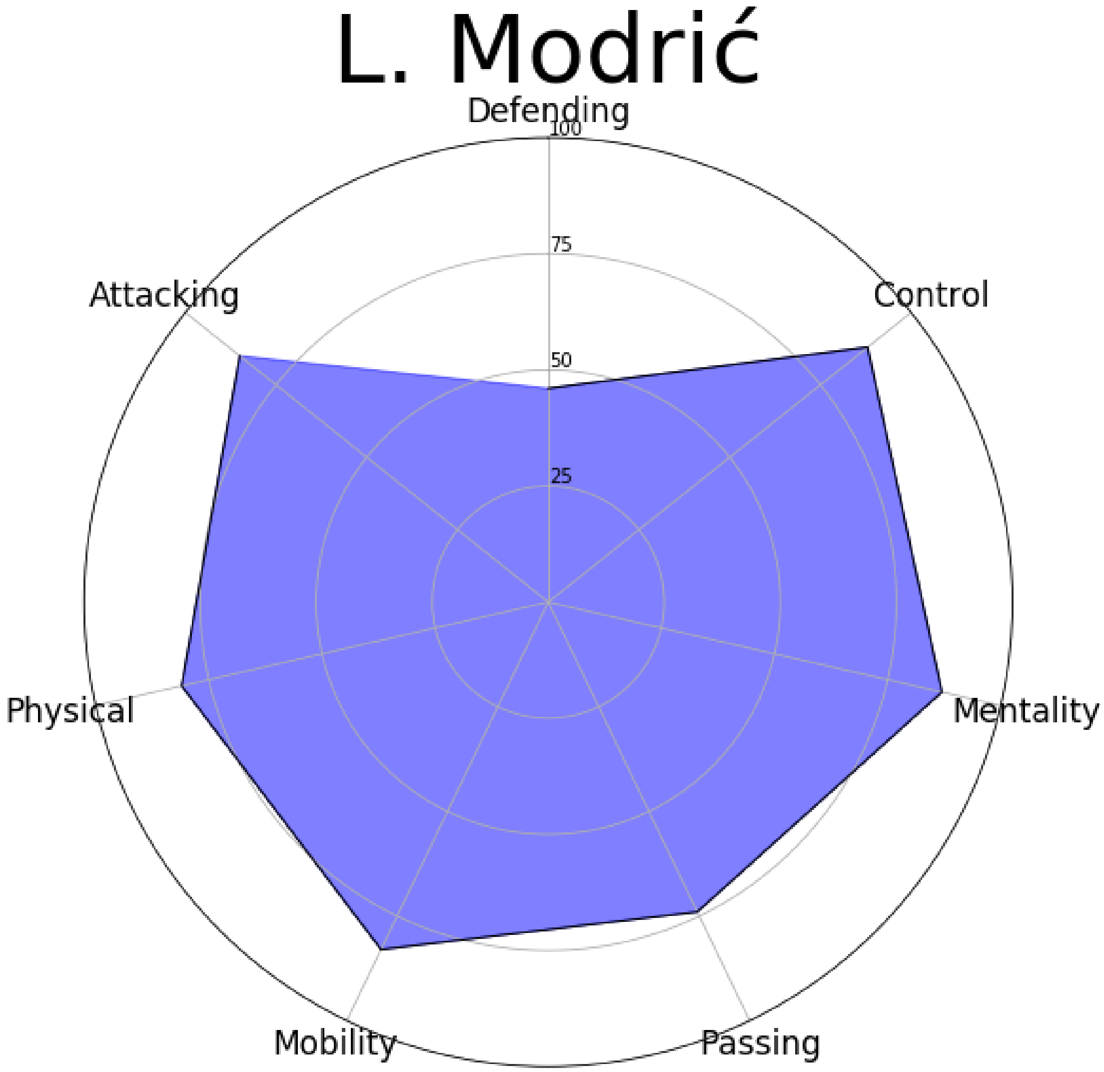

```
In [28]: graphPolarGeneral(1)
```




```
In [29]: graphPolarGeneral(4)
```




```
In [30]: graphPolarGeneral(6)
```



For scouts, when they look in detail of what a player's playstyle is like, they may refer to this easily.

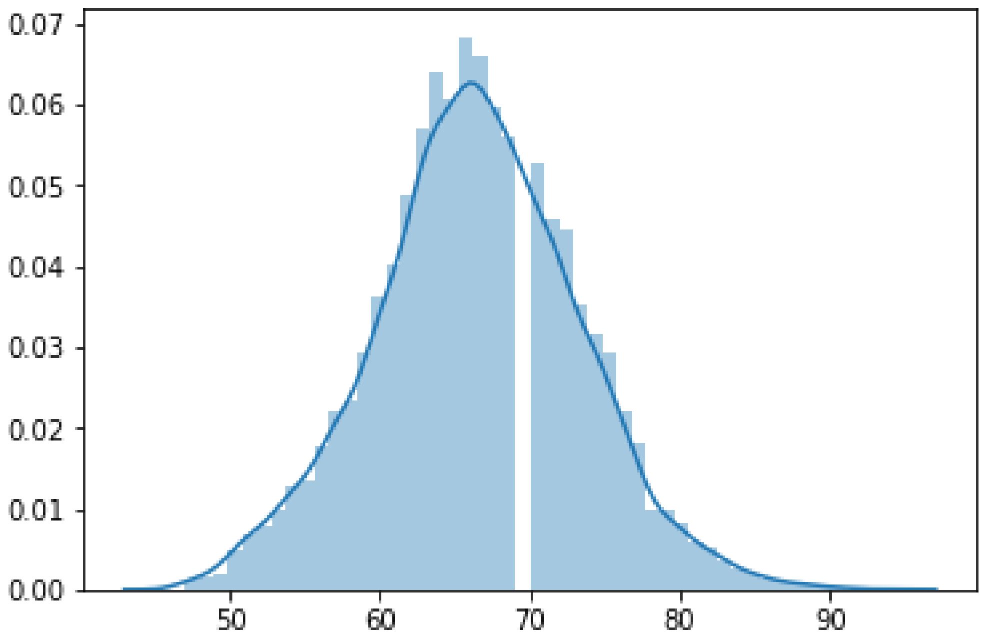
0.1 Now, we look at the data from a higher level: a managerial point of view.

```
In [18]: players = dataset_General[['Name', 'Age','Overall', 'Potential',
                 'Value', 'Crossing', 'Finishing', 'HeadingAccuracy',
                 'ShortPassing', 'Volleys', 'Dribbling', 'Curve', 'FKAccuracy',
                 'LongPassing', 'BallControl', 'Acceleration', 'SprintSpeed', 'Agility',
                 'Reactions', 'Balance', 'ShotPower', 'Jumping', 'Stamina', 'Strength',
                 'LongShots', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
                 'Penalties', 'Composure', 'Marking', 'StandingTackle', 'SlidingTackle',
                 'WeightKG', 'HeightCM']]
```

```
In [60]: # The distribution of player performances
```

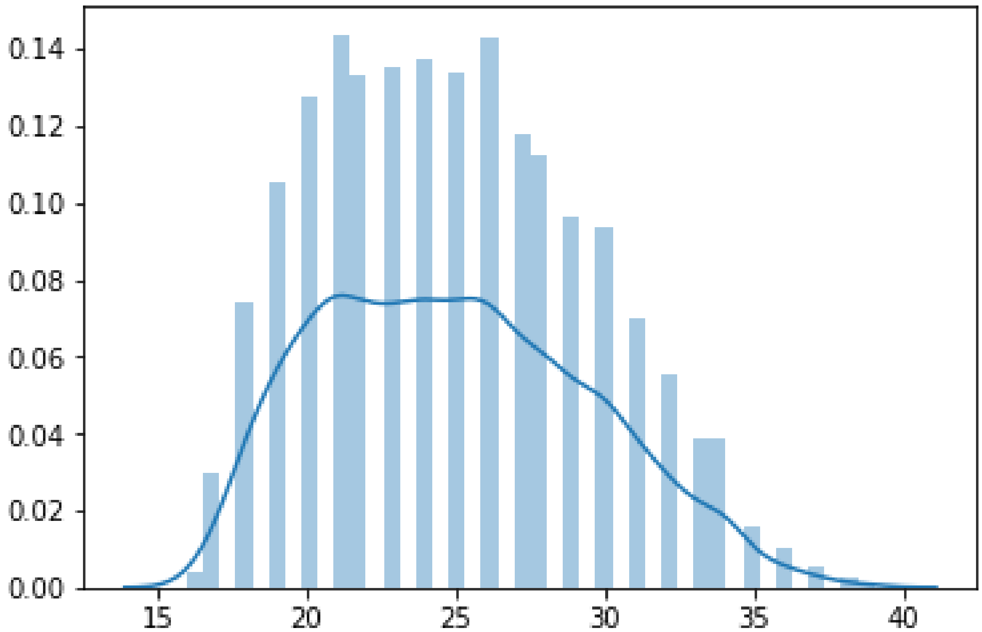
```
sns_plot = sns.distplot(players[['Overall']])
sns_plot.figure.savefig("OverallDist.png")
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



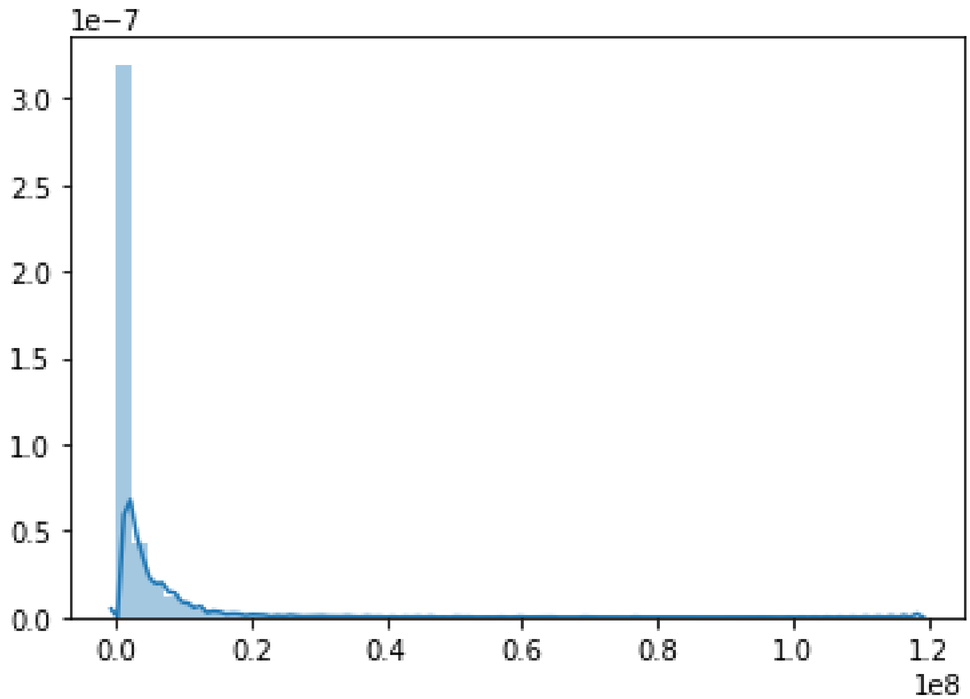

```
In [61]: # The distribution of player Age
         sns_plot = sns.distplot(players[['Age']])
         sns_plot.figure.savefig("AgeDist.png")
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




```
In [62]: # The distribution of player value
         sns_plot = sns.distplot(players[['Value']].astype('float64'))
         sns_plot.figure.savefig("ValueDist.png")
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [76]:
```
#Find the correlation
corr = players[['Age', 'Crossing', 'Finishing', 'HeadingAccuracy',
       'ShortPassing', 'Volleys', 'Dribbling', 'Curve', 'FKAccuracy',
       'LongPassing', 'BallControl', 'Acceleration', 'SprintSpeed', 'Agility',
       'Reactions', 'Balance', 'ShotPower', 'Jumping', 'Stamina', 'Strength',
       'LongShots', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Penalties', 'Composure', 'Marking', 'StandingTackle', 'SlidingTackle',
       'WeightKG', 'HeightCM']].corrwith(players['Overall'])
print(corr.sort_values(ascending=False))
```

```
Reactions          0.848950
Composure          0.803705
ShortPassing       0.724298
BallControl        0.720986
LongPassing        0.585774
ShotPower          0.565120
Vision             0.525972
Dribbling          0.518555
Curve              0.503961
LongShots          0.503115
Crossing           0.497384
HeadingAccuracy    0.468149
Stamina            0.462111
Age                0.457102
```

```
FKAccuracy         0.456814
Aggression         0.454796
Volleys            0.452194
Positioning        0.440056
Penalties          0.390919
Finishing          0.373892
Strength           0.342113
Interceptions      0.334736
Marking            0.307288
StandingTackle     0.265473
Agility            0.244509
Jumping            0.228689
SlidingTackle      0.225215
WeightKG           0.185195
SprintSpeed        0.170393
Acceleration       0.151057
HeightCM           0.067842
Balance            0.059761
dtype: float64
```

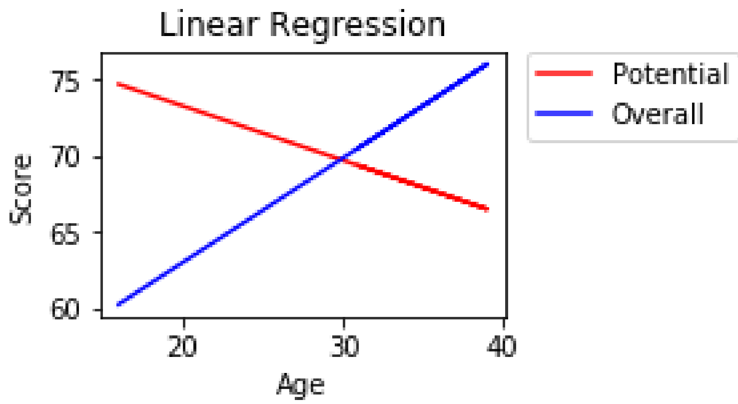
```
In [46]: #Find the correlation
         corr = players[['Age', 'Crossing', 'Finishing', 'HeadingAccuracy',
                'ShortPassing', 'Volleys', 'Dribbling', 'Curve', 'FKAccuracy',
                'LongPassing', 'BallControl', 'Acceleration', 'SprintSpeed', 'Agility',
                'Reactions', 'Balance', 'ShotPower', 'Jumping', 'Stamina', 'Strength',
                'LongShots', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
                'Penalties', 'Composure', 'Marking', 'StandingTackle', 'SlidingTackle',
                    'WeightKG', 'HeightCM']].corrwith(players['Potential'])
         print(corr.sort_values(ascending=False))
```

```
BallControl        0.522945
Reactions          0.505626
ShortPassing       0.495916
Composure          0.470442
Dribbling          0.417096
LongPassing        0.359565
Vision             0.348613
ShotPower          0.331622
Curve              0.306351
LongShots          0.288316
Positioning        0.274038
Crossing           0.273398
Volleys            0.268384
Finishing          0.253364
FKAccuracy         0.234780
SprintSpeed        0.233360
Penalties          0.229240
```

```
Acceleration        0.228676
HeadingAccuracy     0.225146
Agility             0.207838
Stamina             0.206068
Aggression          0.156844
Marking             0.146611
Interceptions       0.131898
StandingTackle      0.121881
Balance             0.116388
SlidingTackle       0.104758
Jumping             0.066395
Strength            0.050525
WeightKG            0.015979
HeightCM            0.012819
Age                -0.266158
dtype: float64
```

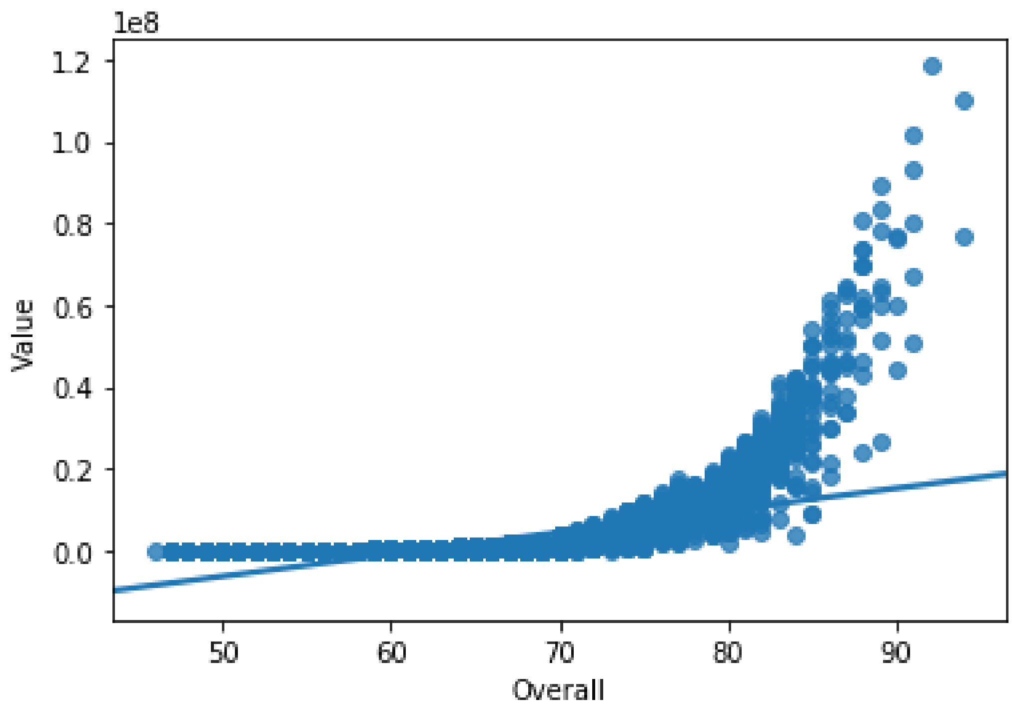
The correlation matrix shows that some of the most important features regarding to a player's performance are: Ball Control, Reactions, Short Passing, Composure,Dribbling, Vision and Long passing. Something worth noticing is that there is a negative correlation between age and potential. While this doesn't imply causal relationship, we can infer that with the increase of age, the potential decreases. Using a regression to confirm:

```
In [69]: import matplotlib.patches as mpatches
         from sklearn.linear_model import LinearRegression
         player_age = players[['Age','Potential']]
         y = player_age[['Potential']].values.ravel()
         X = player_age[player_age.columns.drop('Potential')]
         model1 = LinearRegression()
         model1.fit(X,y)
         y_pred1 = model1.predict(X)
         player_age = players[['Age','Overall']]
         y = player_age[['Overall']].values.ravel()
         X = player_age[player_age.columns.drop('Overall')]
         model2 = LinearRegression()
         model2.fit(X,y)
         y_pred2 = model2.predict(X)
         plt.subplot(223)
         plt.plot(X, y_pred1, color = 'red',label="Potential")
         plt.plot(X, y_pred2, color = 'blue',label = "Overall")
         plt.title('Linear Regression')
         plt.xlabel('Age')
         plt.ylabel('Score')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.savefig('Score~Age.png')

         plt.show()
```



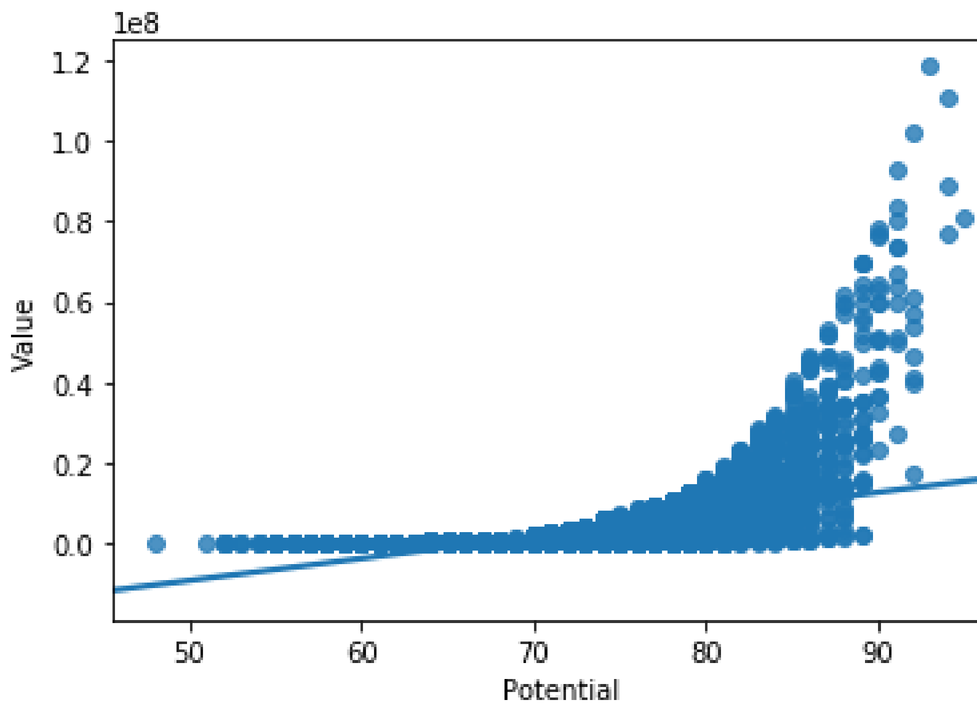

```
In [80]: # The relationship between Overall and value
         sns_plot = sns.regplot(y = 'Value',x="Overall", data=players)
         sns_plot.figure.savefig("RegPlot1.png")
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



```
In [81]: # The relationship between Overall and value
         sns_plot = sns.regplot(y = 'Value',x="Potential", data=players)
         sns_plot.figure.savefig("RegPlot2.png")
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



This shows that, in general, the higher the player's overall score is, the more expensive he would be. However, it is also worth noticing that there are several players who have a reletively high performance but not that high salary. From the polt, I think it would fit a ploynomial regression.

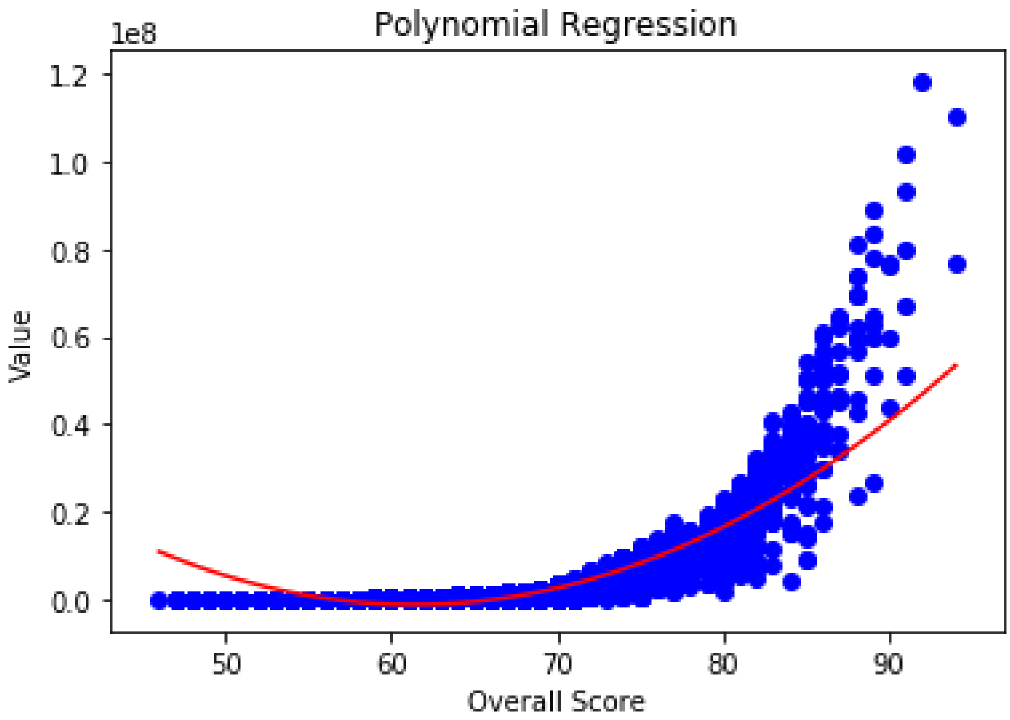
```
In [37]: #then process the data into X and y form
         player_score = players[['Value','Overall']]
         y = player_score[['Value']].values.ravel()
         X = player_score[player_score.columns.drop('Value')]
```

```
In [22]: from sklearn.model_selection import GridSearchCV
         from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
         #Train-test

         param_test = {
```

```
              'poly__degree':range(1,3)
          }

estimator = Pipeline([('poly', PolynomialFeatures()),
                      ('linear', LinearRegression(fit_intercept=False))])
gsearch = GridSearchCV(estimator , param_grid = param_test, cv=10,scoring='neg_mean_squ
gsearch.fit(X,y)
gsearch.best_params_, gsearch.best_score_
print('best score is:',str(gsearch.best_score_))
print('best params are:',str(gsearch.best_params_))
```

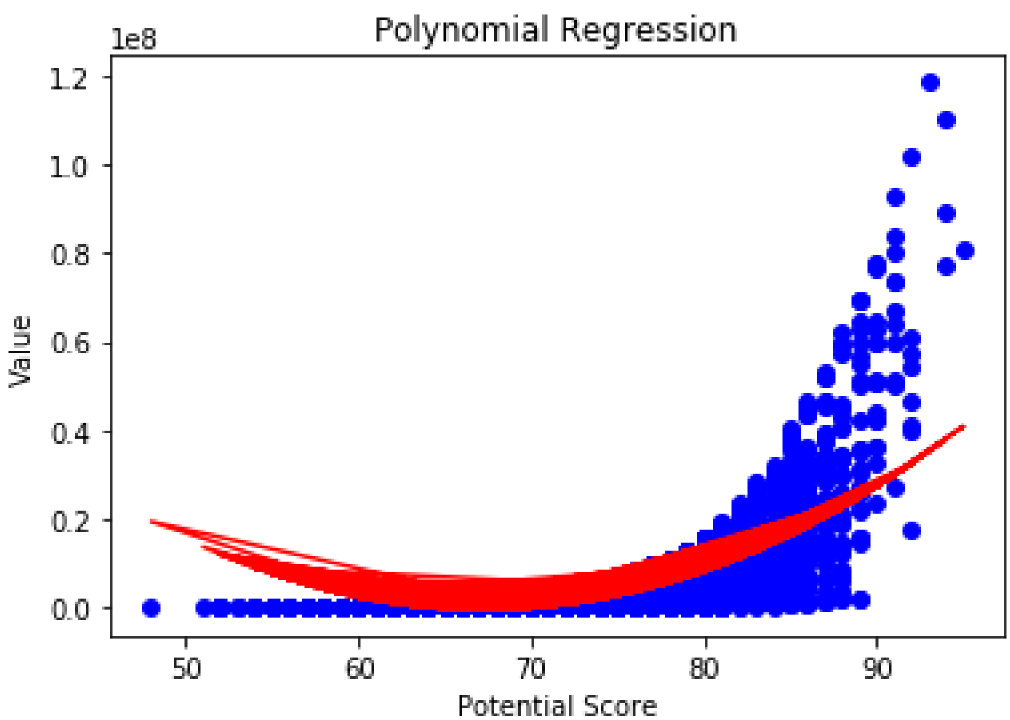
```
best score is: -25786097142084.973
best params are: {'poly__degree': 2}
```

In [39]:
```
# Visualising the Polynomial Regression results
y = player_score[['Value']].values.ravel()
X = player_score[player_score.columns.drop('Value')]
model = Pipeline([('poly', PolynomialFeatures(degree = 2)),
                      ('linear', LinearRegression(fit_intercept=False))])
model.fit(X,y)
y_pred = model.predict(X)

plt.plot(X, y_pred, color = 'red')
plt.scatter(X, y, color = 'blue')
plt.title('Polynomial Regression')
plt.xlabel('Overall Score')
plt.ylabel('Value')
plt.savefig('Value~OveallScore.png')
plt.show()
```




```
In [32]: #then process the data into X and y form
         player_score = players[['Value','Potential']]
         y = player_score[['Value']].values.ravel()
         X = player_score[player_score.columns.drop('Value')]

In [30]: from sklearn.model_selection import GridSearchCV
         from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
         param_test = {
                 'poly__degree':range(1,3)
             }

         estimator = Pipeline([('poly', PolynomialFeatures()),
                              ('linear', LinearRegression(fit_intercept=False))])
         gsearch = GridSearchCV(estimator , param_grid = param_test, cv=10,scoring='neg_mean_squ
         gsearch.fit(X,y)
         gsearch.best_params_, gsearch.best_score_
         print('best score is:',str(gsearch.best_score_))
         print('best params are:',str(gsearch.best_params_))

best score is: -31385312630252.66
best params are: {'poly__degree': 2}
```

```
In [45]: player_score = players[['Value','Potential']]
         y = player_score[['Value']].values.ravel()
         X = player_score[player_score.columns.drop('Value')]

         # Visualising the Polynomial Regression results
         model = Pipeline([('poly', PolynomialFeatures(degree = 2)),
                           ('linear', LinearRegression(fit_intercept=False))])
         model.fit(X,y)
         y_pred = model.predict(X)
         plt.scatter(X,y,color='blue')
         plt.plot(X, y_pred, color = 'red')
         plt.title('Polynomial Regression')
         plt.xlabel('Potential Score')
         plt.ylabel('Value')
         plt.savefig('Value~PotentialScore.png')
         plt.show()
```



As we can see, the polynomial regression with degree 2 is the best fit. According to our plots, we would like to identify the players that have overall/potential score over 80 and are below the respective predicted value. Because we want to evaluate a player based on both overall and potential, we now use both variables.

```
In [56]: player_score = players[['Value','Overall','Name','Potential']]
         y = player_score[['Value']].values.ravel()
         X = player_score[player_score.columns.drop('Value').drop('Name')]
```

In [57]:
```
param_test = {
                'poly__degree':range(1,3)
            }

estimator = Pipeline([('poly', PolynomialFeatures()),
                      ('linear', LinearRegression(fit_intercept=False))])
gsearch = GridSearchCV(estimator , param_grid = param_test, cv=10,scoring='neg_mean_squ
gsearch.fit(X,y)
gsearch.best_params_, gsearch.best_score_
print('best score is:',str(gsearch.best_score_))
print('best params are:',str(gsearch.best_params_))
```

```
best score is: -19676078440368.617
best params are: {'poly__degree': 2}
```

In [79]:
```
model = Pipeline([('poly', PolynomialFeatures(degree = 2)),
                  ('linear', LinearRegression(fit_intercept=False))])
model.fit(X,y)
y_pred = model.predict(X)
A = player_score[['Overall']].values.ravel()
B = player_score[['Potential']].values.ravel()
lst = []
for i in range(y.size):
    if (A[i] >= 80 or B[i] >= 80):
        if y_pred[i] >= y[i]:
            lst.append(i)
listName = []
for i in lst:
    listName.append(player_score[['Name']].values.ravel()[i])
print(listName)
```

```
['Dani Sandoval', 'B. Adekanye', 'A. Wilson', 'I. Sauter']
```

In [ ]: