In [53]:
```
# Predicting players rating
# In this project you are going to predict the overall rating of soccer player
based on their attributes such as
# 'crossing', 'finishing etc.
# The dataset you are going to use is from European Soccer Database(https://ww
w.kaggle.com/hugomathien/soccer) has more
# than 25,000 matches and more than 10,000 players for European professional s
occer seasons from 2008 to 2016.

# About the Dataset
# The ultimate Soccer database for data analysis and machine learning The data
set comes in the form of an SQL database
# and contains statistics of about 25,000 football matches, from the top footb
all league of 11 European Countries.
# It covers seasons from 2008 to 2016 and contains match statistics (i.e: scor
es, corners, fouls etc...) as well as the
# team formations, with player names and a pair of coordinates to indicate the
ir position on the pitch. +25,000
# matches +10,000 players 11 European Countries with their lead championship S
easons 2008 to 2016 Players and
# Teams' attributes* sourced from EA Sports' FIFA video game series, including
the weekly updates Team line up with
# squad formation (X, Y coordinates) Betting odds from up to 10 providers Deta
iled match events (goal types, possession,
# corner, cross, fouls, cards etc...) for +10,000 matches The dataset also has
a set of about 35 statistics for each
# player, derived from EA Sports' FIFA video games. It is not just the stats t
hat come with a new version of the game
# but also the weekly updates. So for instance if a player has performed poorl
y over a period of time and his stats get
# impacted in FIFA, you would normally see the same in the dataset.
```

In [1]:
```
import sqlite3
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Create your connection.
cnx = sqlite3.connect('database.sqlite')
df = pd.read_sql_query("SELECT * FROM Player_Attributes", cnx)
```

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

Out[3]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attackin
0	1	218353	505942	2016-02-18 00:00:00	67.0	71.0	right	
1	2	218353	505942	2015-11-19 00:00:00	67.0	71.0	right	
2	3	218353	505942	2015-09-21 00:00:00	62.0	66.0	right	
3	4	218353	505942	2015-03-20 00:00:00	61.0	65.0	right	
4	5	218353	505942	2007-02-22 00:00:00	61.0	65.0	right	

5 rows × 42 columns

In [5]:
```
df.shape
```

Out[5]:
```
(183978, 42)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183978 entries, 0 to 183977
Data columns (total 42 columns):
id                     183978 non-null int64
player_fifa_api_id     183978 non-null int64
player_api_id          183978 non-null int64
date                   183978 non-null object
overall_rating         183142 non-null float64
potential              183142 non-null float64
preferred_foot         183142 non-null object
attacking_work_rate    180748 non-null object
defensive_work_rate    183142 non-null object
crossing               183142 non-null float64
finishing              183142 non-null float64
heading_accuracy       183142 non-null float64
short_passing          183142 non-null float64
volleys                181265 non-null float64
dribbling              183142 non-null float64
curve                  181265 non-null float64
free_kick_accuracy     183142 non-null float64
long_passing           183142 non-null float64
ball_control           183142 non-null float64
acceleration           183142 non-null float64
sprint_speed           183142 non-null float64
agility                181265 non-null float64
reactions              183142 non-null float64
balance                181265 non-null float64
shot_power             183142 non-null float64
jumping                181265 non-null float64
stamina                183142 non-null float64
strength               183142 non-null float64
long_shots             183142 non-null float64
aggression             183142 non-null float64
interceptions          183142 non-null float64
positioning            183142 non-null float64
vision                 181265 non-null float64
penalties              183142 non-null float64
marking                183142 non-null float64
standing_tackle        183142 non-null float64
sliding_tackle         181265 non-null float64
gk_diving              183142 non-null float64
gk_handling            183142 non-null float64
gk_kicking             183142 non-null float64
gk_positioning         183142 non-null float64
gk_reflexes            183142 non-null float64
dtypes: float64(35), int64(3), object(4)
memory usage: 56.1+ MB
```

In [7]:
```
df = df.dropna()
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180354 entries, 0 to 183977
Data columns (total 42 columns):
id                     180354 non-null int64
player_fifa_api_id     180354 non-null int64
player_api_id          180354 non-null int64
date                   180354 non-null object
overall_rating         180354 non-null float64
potential              180354 non-null float64
preferred_foot         180354 non-null object
attacking_work_rate    180354 non-null object
defensive_work_rate    180354 non-null object
crossing               180354 non-null float64
finishing              180354 non-null float64
heading_accuracy       180354 non-null float64
short_passing          180354 non-null float64
volleys                180354 non-null float64
dribbling              180354 non-null float64
curve                  180354 non-null float64
free_kick_accuracy     180354 non-null float64
long_passing           180354 non-null float64
ball_control           180354 non-null float64
acceleration           180354 non-null float64
sprint_speed           180354 non-null float64
agility                180354 non-null float64
reactions              180354 non-null float64
balance                180354 non-null float64
shot_power             180354 non-null float64
jumping                180354 non-null float64
stamina                180354 non-null float64
strength               180354 non-null float64
long_shots             180354 non-null float64
aggression             180354 non-null float64
interceptions          180354 non-null float64
positioning            180354 non-null float64
vision                 180354 non-null float64
penalties              180354 non-null float64
marking                180354 non-null float64
standing_tackle        180354 non-null float64
sliding_tackle         180354 non-null float64
gk_diving              180354 non-null float64
gk_handling            180354 non-null float64
gk_kicking             180354 non-null float64
gk_positioning         180354 non-null float64
gk_reflexes            180354 non-null float64
dtypes: float64(35), int64(3), object(4)
memory usage: 56.4+ MB
```

In [9]:
```
# Feature selection - Since we have about 40 different features, we can run so
me feature selection algorithms to reduce
# the size of our featureset.
from sklearn.preprocessing import scale
from sklearn.feature_selection import RFE
```

In [10]:
```
df_new = df.copy()
```

In [11]:
```
# Most of our data is numeric, with few exceptions having floating data types. In the subsequent prediction analysis
# we'll only concern ourself with the integer numerics, but there is obviously potential gains to be made by
# incorporating the qualitative data (i.e. player position).
df_new = df_new.select_dtypes(["int64","float64"])
```

In [12]:
```
df_new.shape
```

Out[12]:
```
(180354, 38)
```

In [13]:
```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180354 entries, 0 to 183977
Data columns (total 38 columns):
id                     180354 non-null int64
player_fifa_api_id     180354 non-null int64
player_api_id          180354 non-null int64
overall_rating         180354 non-null float64
potential              180354 non-null float64
crossing               180354 non-null float64
finishing              180354 non-null float64
heading_accuracy       180354 non-null float64
short_passing          180354 non-null float64
volleys                180354 non-null float64
dribbling              180354 non-null float64
curve                  180354 non-null float64
free_kick_accuracy     180354 non-null float64
long_passing           180354 non-null float64
ball_control           180354 non-null float64
acceleration           180354 non-null float64
sprint_speed           180354 non-null float64
agility                180354 non-null float64
reactions              180354 non-null float64
balance                180354 non-null float64
shot_power             180354 non-null float64
jumping                180354 non-null float64
stamina                180354 non-null float64
strength               180354 non-null float64
long_shots             180354 non-null float64
aggression             180354 non-null float64
interceptions          180354 non-null float64
positioning            180354 non-null float64
vision                 180354 non-null float64
penalties              180354 non-null float64
marking                180354 non-null float64
standing_tackle        180354 non-null float64
sliding_tackle         180354 non-null float64
gk_diving              180354 non-null float64
gk_handling            180354 non-null float64
gk_kicking             180354 non-null float64
gk_positioning         180354 non-null float64
gk_reflexes            180354 non-null float64
dtypes: float64(35), int64(3)
memory usage: 53.7 MB
```

In [14]:
```
X = df_new.drop('overall_rating',axis=1).values
y = df_new['overall_rating'].values.ravel()
from sklearn.preprocessing import scale
X = scale(X)
```

In [15]:
```
df_1 = df_new.drop('overall_rating',axis=1)
```

In [16]:
```
X.shape
```

Out[16]:
```
(180354, 37)
```

In [17]:
```
y.shape
```

Out[17]:
```
(180354,)
```

In [18]:
```
# Feature Selection using RFE Scikit Library

lm = LinearRegression()

rfe = RFE(lm, n_features_to_select = 10)
rfe_fit = rfe.fit(X, y)
features = []
for feat in df_1.columns[rfe_fit.support_]:
    print(feat)
    features.append(feat)
```

```
player_api_id
potential
heading_accuracy
short_passing
ball_control
reactions
strength
gk_diving
gk_kicking
gk_positioning
```

In [19]:
```
features
```

Out[19]:
```
['player_api_id',
 'potential',
 'heading_accuracy',
 'short_passing',
 'ball_control',
 'reactions',
 'strength',
 'gk_diving',
 'gk_kicking',
 'gk_positioning']
```

In [20]:
```
# Using Statsmodel to illustrate the summary results
lm = LinearRegression()

rfe = RFE(lm, n_features_to_select = 15)
rfe_fit = rfe.fit(X, y)
features = []
for feat in df_1.columns[rfe_fit.support_]:
    print(feat)
    features.append(feat)
```

```
player_fifa_api_id
player_api_id
potential
heading_accuracy
short_passing
ball_control
acceleration
reactions
strength
marking
gk_diving
gk_handling
gk_kicking
gk_positioning
gk_reflexes
```

In [21]:
```
df_optm = df_new[features]
```

In [22]:
```
df_optm.shape
```

Out[22]:
```
(180354, 15)
```

In [23]:
```
# Using Statsmodels for analysing the impact of attribute potential on the pla
yer rating
import statsmodels.formula.api as sm
model1 = sm.OLS(df_new['overall_rating'],df_new['potential'])
result1 = model1.fit()
print(result1.summary())
```

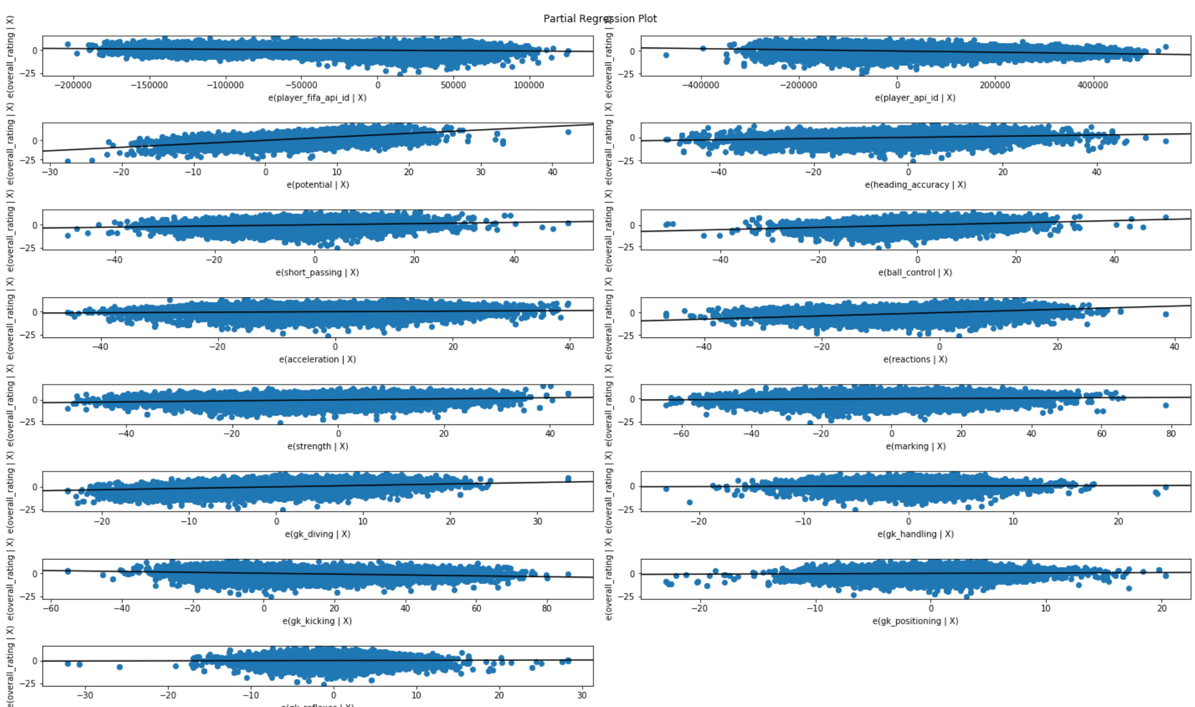
```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:         overall_rating   R-squared:                       0.99
6
Model:                            OLS   Adj. R-squared:                  0.99
6
Method:                 Least Squares   F-statistic:                 4.062e+0
7
Date:                Wed, 06 Mar 2019   Prob (F-statistic):              0.0
0
Time:                        12:26:23   Log-Likelihood:            -5.3063e+0
5
No. Observations:              180354   AIC:                         1.061e+0
6
Df Residuals:                  180353   BIC:                         1.061e+0
6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
potential      0.9331      0.000   6373.716      0.000       0.933       0.93
3
==============================================================================
=
Omnibus:                    30950.684   Durbin-Watson:                   0.37
5
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            54474.35
7
Skew:                          -1.111   Prob(JB):                        0.0
0
Kurtosis:                       4.520   Cond. No.                        1.0
0
==============================================================================
=

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

In [24]:
```
# Using Statsmodels for analysing the impact of all attribute on the player ra
ting
X_new = df_new[features].values
model = sm.OLS(df_new['overall_rating'],df_new[features])
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:          overall_rating   R-squared:                       0.99
9
Model:                             OLS   Adj. R-squared:                  0.99
9
Method:                  Least Squares   F-statistic:                 8.035e+0
6
Date:                 Wed, 06 Mar 2019   Prob (F-statistic):               0.0
0
Time:                         12:27:20   Log-Likelihood:            -4.3282e+0
5
No. Observations:               180354   AIC:                         8.657e+0
5
Df Residuals:                   180339   BIC:                         8.658e+0
5
Df Model:                           15
Covariance Type:             nonrobust
==============================================================================
=========
                        coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
player_fifa_api_id -8.998e-06   1.44e-07    -62.432      0.000   -9.28e-06
-8.72e-06
player_api_id      -6.663e-06   6.01e-08   -110.802      0.000   -6.78e-06
-6.55e-06
potential              0.4525      0.001    327.126      0.000       0.450
0.455
heading_accuracy       0.0643      0.001     93.120      0.000       0.063
0.066
short_passing          0.0646      0.001     59.132      0.000       0.062
0.067
ball_control           0.1317      0.001    104.742      0.000       0.129
0.134
acceleration           0.0299      0.001     43.957      0.000       0.029
0.031
reactions              0.1811      0.001    175.817      0.000       0.179
0.183
strength               0.0600      0.001     91.087      0.000       0.059
0.061
marking                0.0187      0.000     49.000      0.000       0.018
0.019
gk_diving              0.1627      0.001    117.671      0.000       0.160
0.165
gk_handling            0.0254      0.002     13.865      0.000       0.022
0.029
gk_kicking            -0.0479      0.001    -83.797      0.000      -0.049
-0.047
gk_positioning         0.0437      0.002     23.984      0.000       0.040
0.047
gk_reflexes            0.0179      0.002      9.987      0.000       0.014
0.021
==============================================================================
=
```

```
Omnibus:                    13814.579   Durbin-Watson:                   0.42
5
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            34846.32
8
Skew:                          -0.457   Prob(JB):                         0.0
0
Kurtosis:                       4.950   Cond. No.                    8.42e+0
4
==============================================================================
=

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 8.42e+04. This might indicate that there a
re
strong multicollinearity or other numerical problems.
```

In [25]:
```
# Explanation of the OLS Regression Results :
# Adjusted R-squared indicates that 99.9% of player ratings can be explained by our predictor variable.
# The regression coefficient (coef) represents the change in the dependent variable resulting from a one unit change in
# the predictor variable, all other variables being held constant.
# In our model, a one unit increase in potential increases the rating by 0.4525.
# The standard error measures the accuracy of potential's coefficient by estimating the variation of the coefficient
# if the same test were run on a different sample of our population.
# Our standard error,0.001, is low and therefore appears accurate.
# The p-value means the probability of an 0.4525 increasing in player rating due to a one unit increase in potential
# is 0% , assuming there is no relationship between the two variables.
# A low p-value indicates that the results are statistically significant, that is in general the p-value is less than 0.05.
# The confidence interval is a range within which our coefficient is likely to fall. We can be 95% confident that
# potentials's coefficient will be within our confidence interval, [0.450,0.455].
# Warnings in the Summary provided by statsmodels OLS model
# Multicollinearity: A careful observer would've noticed the warnings produced by our model regarding multicollinearity.
# We have two or more variables telling roughly the same story, overstating the value of each of the predictors.

# REGRESSION PLOTS

import matplotlib.pyplot as plt
import statsmodels.api as sm
%matplotlib inline
fig = plt.figure(figsize=(20,12))
fig = sm.graphics.plot_partregress_grid(result, fig=fig)
```



In [26]:
```
import statsmodels.formula.api as smf
# only include TV and Radio in the model
lm = smf.ols(formula='overall_rating ~  player_fifa_api_id + player_api_id +potential +heading_accuracy +short_passing +ball_control +acceleration +reactions +strength +marking +gk_diving +gk_handling +gk_kicking +gk_positioning +gk_reflexes ', data=df_new).fit()
print('Confidence of the statsmodel for the input data : ',lm.rsquared)
```

```
Confidence of the statsmodel for the input data :  0.8561047414083593
```

In [27]:
```
df_new.head()
```

Out[27]:

	id	player_fifa_api_id	player_api_id	overall_rating	potential	crossing	finishing	heading_accu
0	1	218353	505942	67.0	71.0	49.0	44.0	
1	2	218353	505942	67.0	71.0	49.0	44.0	
2	3	218353	505942	62.0	66.0	49.0	44.0	
3	4	218353	505942	61.0	65.0	48.0	43.0	
4	5	218353	505942	61.0	65.0	48.0	43.0	

5 rows × 38 columns

In [28]:
```
df_optm.head()
```

Out[28]:

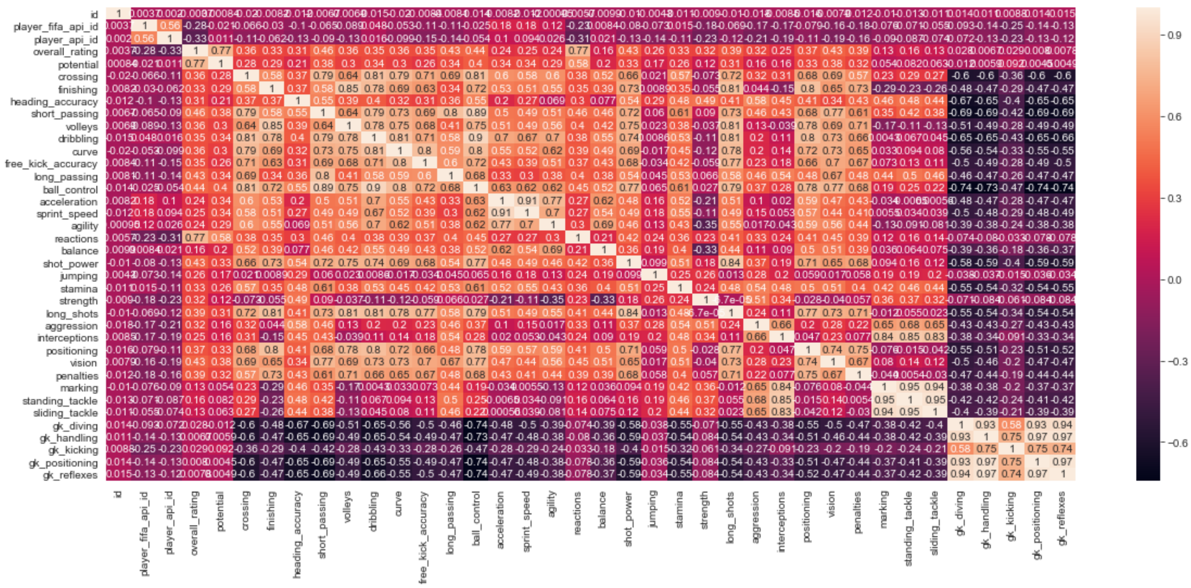
	player_fifa_api_id	player_api_id	potential	heading_accuracy	short_passing	ball_control	acc
0	218353	505942	71.0	71.0	61.0	49.0	
1	218353	505942	71.0	71.0	61.0	49.0	
2	218353	505942	66.0	71.0	61.0	49.0	
3	218353	505942	65.0	70.0	60.0	48.0	
4	218353	505942	65.0	70.0	60.0	48.0	

In [29]:
```
# Data Exploration using visualization
df_corr = df_new.corr()
```

In [30]:
```python
print('The features contain high corelation .We need to remove them first before applying Regression Techniques.')
import seaborn as sns
sns.set_style('whitegrid')
plt.figure(figsize=(20,8))
sns.heatmap(df_corr,annot=True)
```

```
The features contain high corelation .We need to remove them first before applying Regression Techniques.
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x107cef30>
```



In [31]:
```python
#create correlation matrix with absolute values
df_corr = df_new.corr().abs()
#select upper triangle of matrix
up_tri = df_corr.where(np.triu(np.ones(df_corr.shape[1]),k=1).astype(np.bool))

#find all the features which have a correlation > 0.75 with other features.
corr_features = [ column for column in up_tri.columns if any(up_tri[column]>0.75)]

#Print Correlated features
print(corr_features)
```

```
['potential', 'short_passing', 'volleys', 'dribbling', 'curve', 'free_kick_accuracy', 'long_passing', 'ball_control', 'sprint_speed', 'agility', 'reactions', 'shot_power', 'long_shots', 'positioning', 'vision', 'penalties', 'marking', 'standing_tackle', 'sliding_tackle', 'gk_handling', 'gk_positioning', 'gk_reflexes']
```

In [32]:
```
#Drop Correlated Features
df_no_corr = df_new.drop(corr_features,axis=1)
df_no_corr.head()
```

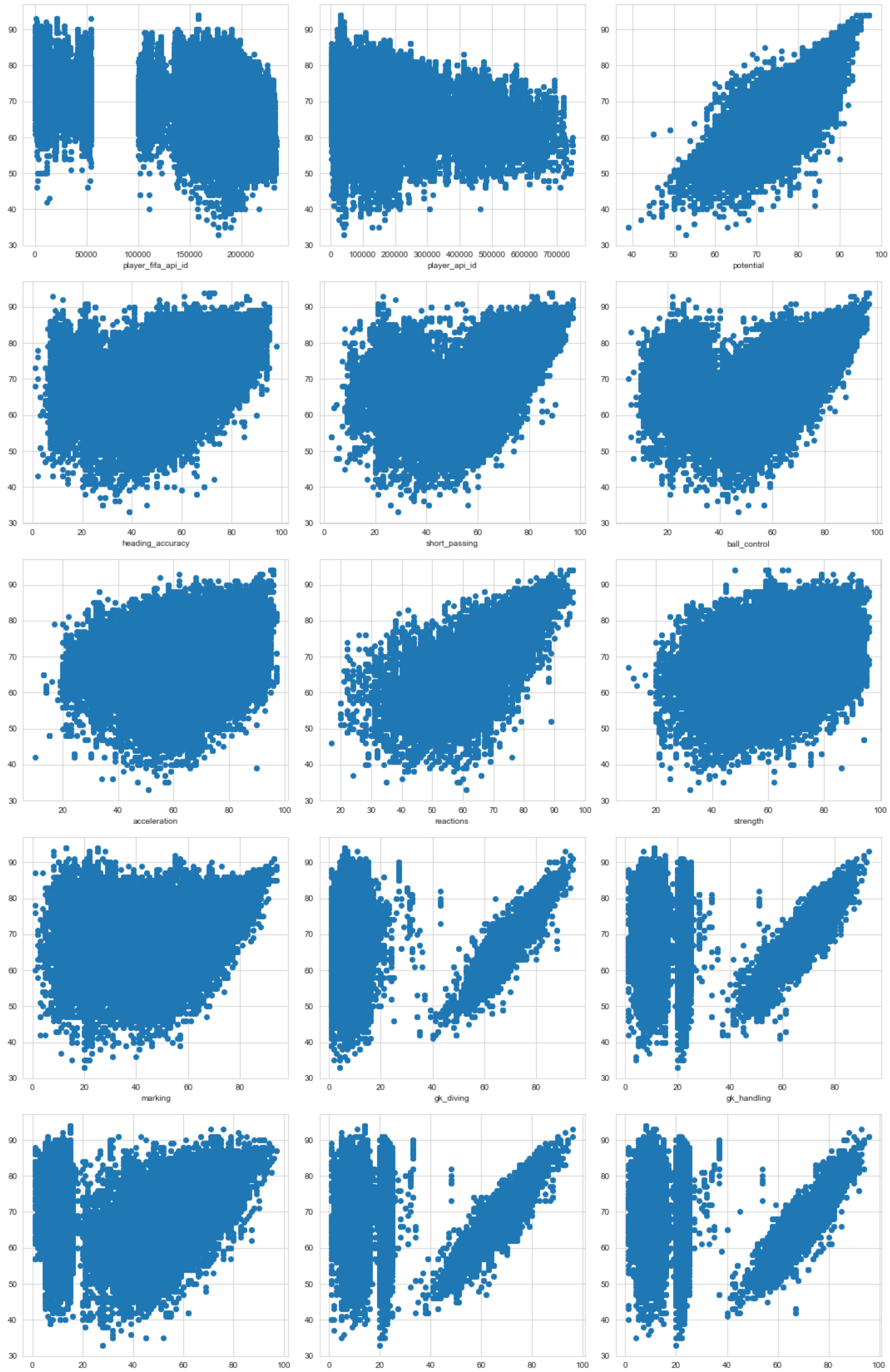
Out[32]:

	id	player_fifa_api_id	player_api_id	overall_rating	crossing	finishing	heading_accuracy	acc
0	1	218353	505942	67.0	49.0	44.0	71.0	
1	2	218353	505942	67.0	49.0	44.0	71.0	
2	3	218353	505942	62.0	49.0	44.0	71.0	
3	4	218353	505942	61.0	48.0	43.0	70.0	
4	5	218353	505942	61.0	48.0	43.0	70.0	

In [33]:
```
# This shows that the feature selection API - sklearn.feature_selection.RFE ha
s resulted in the same feature
# selection for top 15 features selected.
# Quant Features against Rating

len(df_no_corr.columns)
```

Out[33]: 16

In [34]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (15,60))
val = df_optm.shape[1]
for idx in range(val):
    feature = df_optm.columns[idx]
    ax = fig.add_subplot(13,3,idx+1)
    Xtmp = df_optm[feature]
    ax.scatter(Xtmp, y)
    ax.set_xlabel(feature)

plt.tight_layout()
plt.show()
```


player_fifa_api_id
player_api_id
potential
heading_accuracy
short_passing
ball_control
acceleration
reactions
strength
marking
gk_diving
gk_handling
gk_kicking
gk_positioning
gk_reflexes

In [35]:
```
# Split the input data into training and test data
from sklearn.model_selection import train_test_split
#spliting 66.66% for train data and 33.33% for test data.
X_train,X_test,Y_train,Y_test = train_test_split(X,y,test_size=0.33,random_sta
te=0)
print("X_train Shape : ",X_train.shape)
print("X_test Shape : ",X_test.shape)
print("Y_train Shape : ",Y_train.shape)
print("Y_test.shape : ",Y_test.shape)
```

```
X_train Shape :  (120837, 37)
X_test Shape :  (59517, 37)
Y_train Shape :  (120837,)
Y_test.shape :  (59517,)
```

In [36]:
```
# Applying Linear Regression Model
lm = LinearRegression()
lm.fit(X_train, Y_train)# train the model
```

Out[36]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

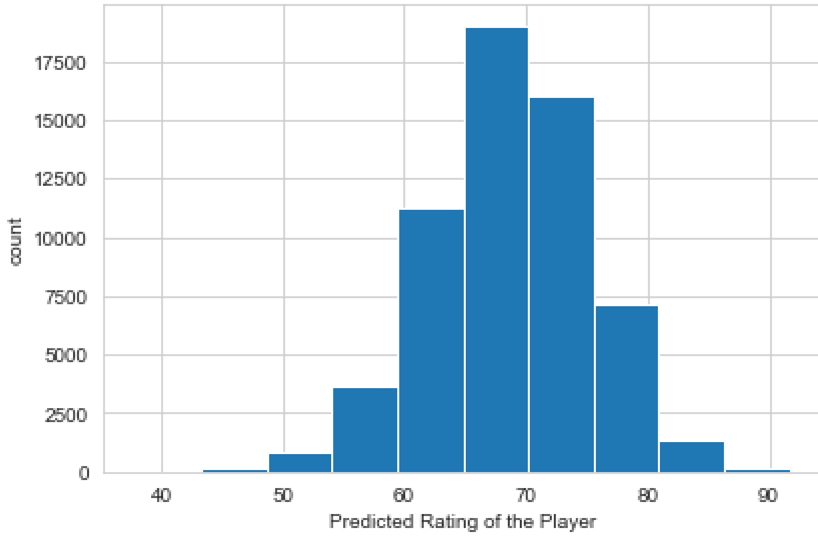
In [37]:
```
# Perform Prediction using Linear Regression Model
Y_pred = lm.predict(X_test) # predict the prices based on the test data
Y_pred
```

Out[37]:
```
array([73.74672865, 68.65401302, 67.22132467, ..., 71.6247244 ,
       64.76934151, 62.18654694])
```

In [38]:
```
print("The variance score of the LinearRegression model is  : ",lm.score(X_tes
t,Y_test))
print('Since variance score is near about 1 it seems to be a perfect predictio
n')
```

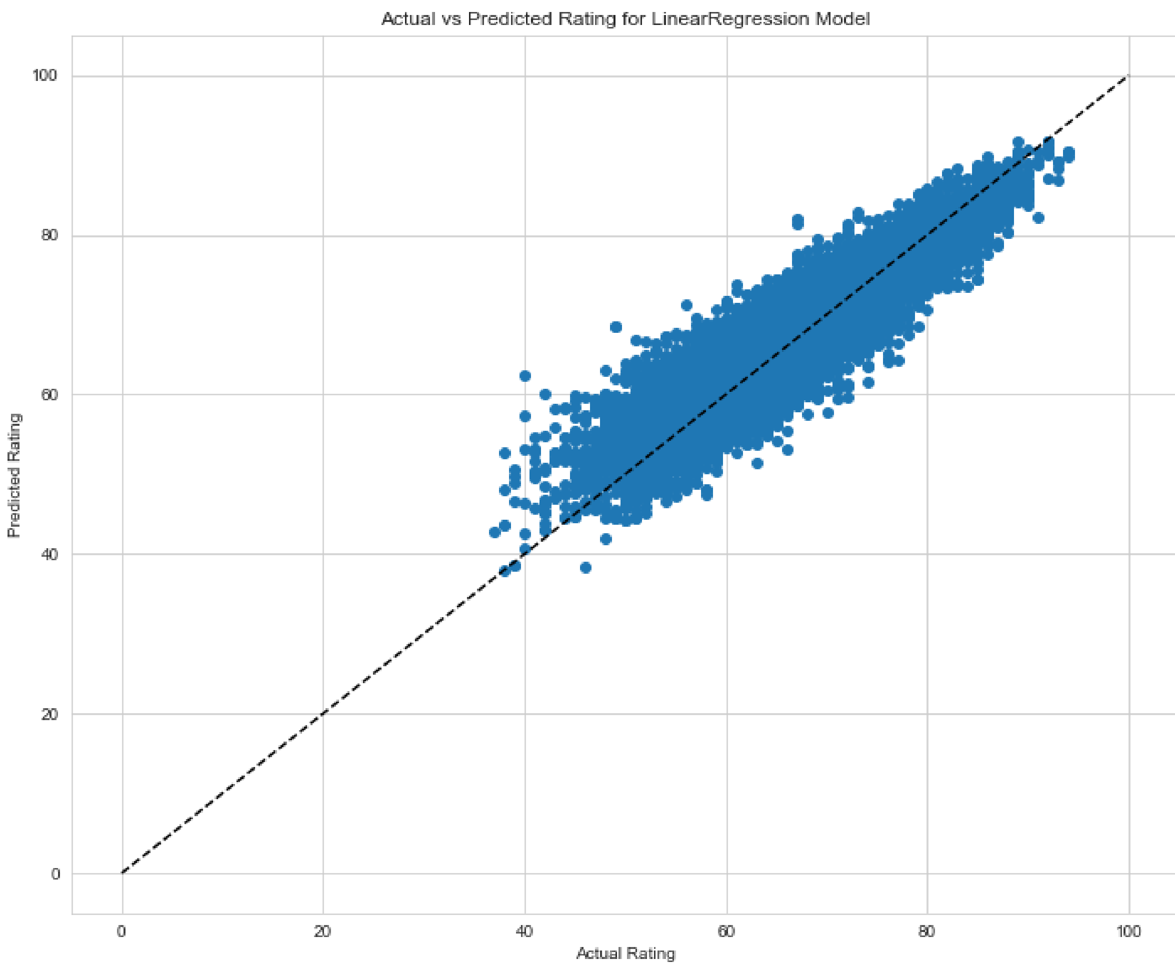
```
The variance score of the LinearRegression model is  :  0.8593275836597539
Since variance score is near about 1 it seems to be a perfect prediction
```

In [39]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
plt.hist(Y_pred)
plt.xlabel('Predicted Rating of the Player')
plt.ylabel('count')
plt.tight_layout()
```



In [40]:
```
import seaborn as sns
sns.set_style('whitegrid')
plt.figure(figsize=(10, 8))
plt.scatter(Y_test, Y_pred)
plt.plot([0, 100], [0, 100], '--k')
plt.axis('tight')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.tight_layout()
plt.title("Actual vs Predicted Rating for LinearRegression Model")
```

Out[40]:
```
Text(0.5, 1.0, 'Actual vs Predicted Rating for LinearRegression Model')
```



In [41]:
```
# Evaluate Linear Regression Accuracy using Root Mean Square Error
from sklearn.metrics import mean_squared_error
print("Error Rate of the Regression Model : ",sqrt(mean_squared_error(Y_pred,Y
_test)))
```

```
Error Rate of the Regression Model :  2.623473911844569
```

In [42]:
```
# Applying Decision Tree Regressor Model to the input data
regressor  = DecisionTreeRegressor(max_depth=20)
regressor.fit(X_train, Y_train)
```

Out[42]:
```
DecisionTreeRegressor(criterion='mse', max_depth=20, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

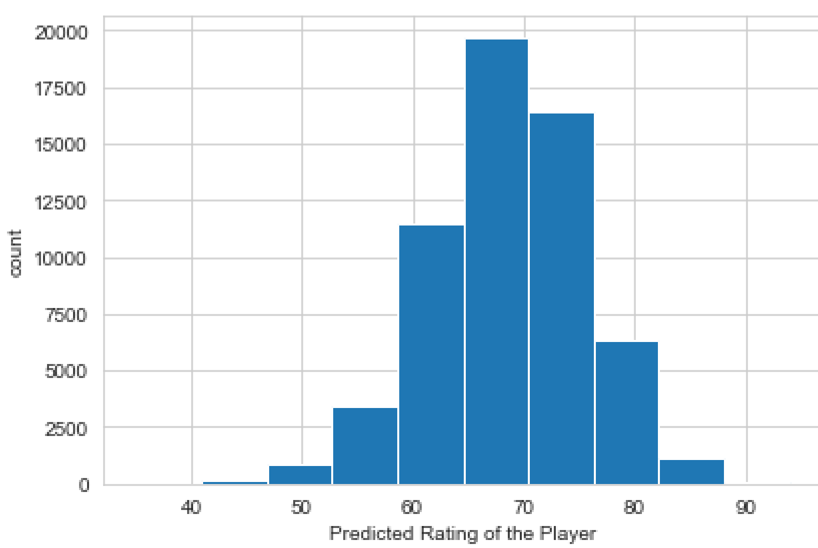
In [43]:
```
# Perform Prediction using Decision Tree Regressor

Y_pred = regressor.predict(X_test)
Y_pred
```

Out[43]:
```
array([76.        , 72.        , 67.        , ..., 71.        ,
       62.        , 61.78571429])
```

In [44]:
```
print("The variance score of the DecisionTreeRegressor model is  : ",regressor
.score(X_test,Y_test))
print('Since variance score is near about 1 it seems to be a perfect predictio
n')
```

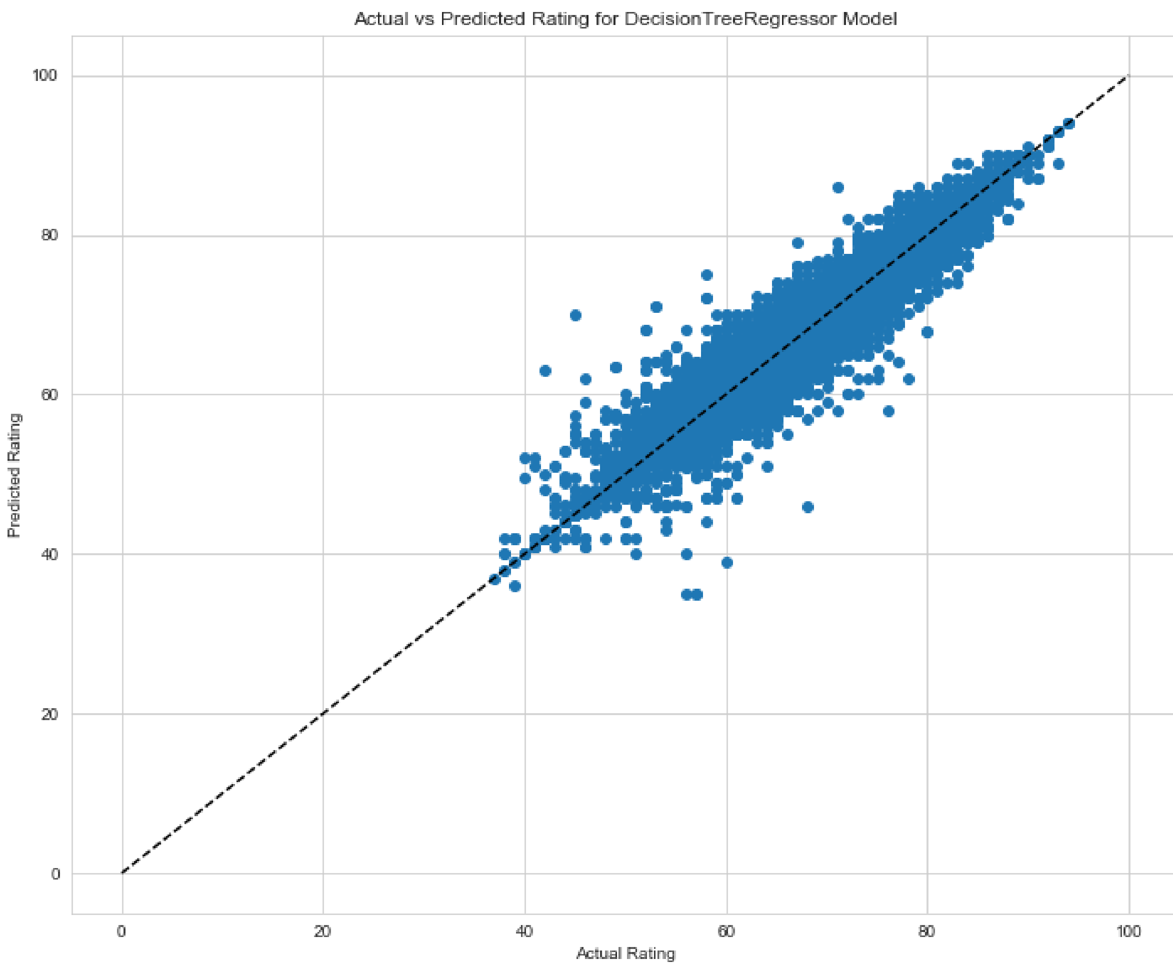
```
The variance score of the DecisionTreeRegressor model is  :  0.95475093587897
91
Since variance score is near about 1 it seems to be a perfect prediction
```

In [45]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
plt.hist(Y_pred)
plt.xlabel('Predicted Rating of the Player')
plt.ylabel('count')
plt.tight_layout()
```



In [46]:
```
plt.figure(figsize=(10, 8))
plt.scatter(Y_test, Y_pred)
plt.plot([0, 100], [0, 100], '--k')
plt.axis('tight')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.tight_layout()
plt.title("Actual vs Predicted Rating for DecisionTreeRegressor Model")
```

Out[46]: `Text(0.5, 1.0, 'Actual vs Predicted Rating for DecisionTreeRegressor Model')`



In [47]:
```
#The mean of the expected target value in test set
Y_test.mean()
```

Out[47]: 68.66518809751835

In [48]:
```
#The mean of the predicted target value in test set ?
Y_pred.mean()
```

Out[48]: 68.65974748152661

In [49]:
```
# Evaluate Linear Regression Accuracy using Root Mean Square Error For DecisionTreeRegressor model
print("Error Rate of the DecisionTreeRegressor Model : ",sqrt(mean_squared_error(Y_pred,Y_test)))
print('The DecisionTreeRegressor Model performs better than the LinearRegression Model as eveident from the error rate')
```

```
Error Rate of the DecisionTreeRegressor Model :  1.487911418908021
The DecisionTreeRegressor Model performs better than the LinearRegression Model as eveident from the error rate
```

In [50]:
```
# Obtaining predictions by cross-validation for the Regression Models

df_optm = df_new.copy()
df_optm['rating'] = y
df_optm.head()
```

Out[50]:

	id	player_fifa_api_id	player_api_id	overall_rating	potential	crossing	finishing	heading_acc
0	1	218353	505942	67.0	71.0	49.0	44.0	
1	2	218353	505942	67.0	71.0	49.0	44.0	
2	3	218353	505942	62.0	66.0	49.0	44.0	
3	4	218353	505942	61.0	65.0	48.0	43.0	
4	5	218353	505942	61.0	65.0	48.0	43.0	

5 rows × 39 columns

In [51]:
```
from sklearn.model_selection import cross_val_predict
X = df_optm.drop('rating',axis=1)
Y = df_optm['rating']
predicted = cross_val_predict(regressor, X, Y, cv=10)
```

In [52]:
```
from sklearn.metrics import accuracy_score
print( "Accuracy Score of the DecisionTreeRegressor Model is  : " ,accuracy_score(Y.astype(int), predicted) )
```

```
Accuracy Score of the DecisionTreeRegressor Model is  :  0.9997615800037704
```

In [62]:
```
# Calculate Error using K-Fold Cross validation
from sklearn.cross_validation import KFold
kfold = KFold(len(df_optm),n_folds=10,shuffle=True,random_state=0)
```

In [65]:
```
from sklearn.metrics import mean_absolute_error
lm = LinearRegression()
mean_abs_error = []
accuracy_score = []
for train,test in kfold:
    x = X.iloc[train]
    y = Y.iloc[train]
    lm.fit(x,y)
    Y_test = Y.iloc[test]
    Y_pred = lm.predict(X.iloc[test])
    mean_abs_error.append(mean_absolute_error(Y_test,Y_pred))
```

In [70]:
```
print('10 Fold Cross Validation Error : {} accuracy score : {}  for LinearRegr
ession Model '.format(np.mean(mean_abs_error),1 - np.mean(mean_abs_error)))
```

```
10 Fold Cross Validation Error : 3.6599463054050l4e-12 accuracy score : 0.999
99999999634  for LinearRegression Model
```

In [71]:
```
from sklearn.metrics import mean_absolute_error
#DR = LinearRegression()
mean_abs_error = []
accuracy_score = []
for train,test in kfold:
    x = X.iloc[train]
    y = Y.iloc[train]
    regressor.fit(x,y)
    Y_test = Y.iloc[test]
    Y_pred = regressor.predict(X.iloc[test])
    mean_abs_error.append(mean_absolute_error(Y_test,Y_pred))
```

In [72]:
```
print('10 Fold Cross Validation Error : {} accuracy score : {}  for DecisionTr
eeRegressor Model '.format(np.mean(mean_abs_error),1 - np.mean(mean_abs_error
)))
```

```
10 Fold Cross Validation Error : 4.435819240365955e-05 accuracy score : 0.999
9556418075963  for DecisionTreeRegressor Model
```

In [ ]:
```
# We have use the below models to predict the player ratings:
# 1) Statsmodels.api.OLS
# 2) LinearRegression
# 3) DecisionTreeRegressor

# Sampling Mechanisms used:
# 1) Test Train Split
# 2) 10 Fold Cross Validation

# Model Estimation mechanisms used:
# 1) Root Mean Squared Error
# 2) 10 Fold Cross Validation error.
```