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://www.kaggle.com/hugomathien/soccer (https://www.kaggle.com/hugomathien/soccer)) has more than 25,000 matches and more than 10,000 players for European professional soccer seasons from 2008 to 2016.

Solution

Importing the required libraries...

In [36]:

```
import sqlite3
import pandas as pd
import statsmodels.api as sm
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import Imputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

Read Data from the Database into pandas

In [37]:

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

Out[37]:

	id	player_fifa_api_id	player_api_id	date	overall_rating	potential	preferred_foot	attacki
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 rows × 42 columns

Basic Information on the dataset

In [38]:

```
# to see the columns and other information on the given data set
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: 59.0+ MB
```

we can see that there are some null values present in the given dataset.So, the next step would be to address them.

In [39]:

```
#Details of categorical variable preferred_foot
df.preferred_foot.value_counts()
```

Out[39]:

```
right    138409
left      44733
Name: preferred_foot, dtype: int64
```

In [40]:

```
#Details of categorical variable attacking work rate
df.attacking_work_rate.value_counts()
```

Out[40]:

```
medium    125070
high       42823
low         8569
None        3639
norm         348
y            106
le           104
stoc          89
Name: attacking_work_rate, dtype: int64
```

In [41]:

```
#Details of categorical variable defencsive work rate
df.defensive_work_rate.value_counts()
```

Out[41]:

```
medium    130846
high       27041
low        18432
_0          2394
o           1550
1            441
ormal        348
2            342
3            258
5            234
7            217
6            197
0            197
9            152
4            116
es           106
ean          104
tocky         89
8             78
Name: defensive_work_rate, dtype: int64
```

In [42]:

```
df.shape
```

Out[42]:

```
(183978, 42)
```

Null Rows analysis in data

In [43]:

```
columns = df.columns
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': columns,'No. Of Missing rows':df.isnull().s
print("The Null/NA missing Data in Basket ball data is : \n")
missing_value_df.sort_values('percent_missing')
```

```
The Null/NA missing Data in Basket ball data is :
```

Out[43]:

	column_name	No. Of Missing rows	No.of rows	percent_missing
id	id	0	183978	0.000000
player_fifa_api_id	player_fifa_api_id	0	183978	0.000000
player_api_id	player_api_id	0	183978	0.000000
date	date	0	183978	0.000000
shot_power	shot_power	836	183142	0.454402
stamina	stamina	836	183142	0.454402
strength	strength	836	183142	0.454402
long_shots	long_shots	836	183142	0.454402
aggression	aggression	836	183142	0.454402
interceptions	interceptions	836	183142	0.454402
penalties	penalties	836	183142	0.454402
reactions	reactions	836	183142	0.454402
marking	marking	836	183142	0.454402
standing_tackle	standing_tackle	836	183142	0.454402
gk_diving	gk_diving	836	183142	0.454402
gk_handling	gk_handling	836	183142	0.454402
gk_kicking	gk_kicking	836	183142	0.454402
positioning	positioning	836	183142	0.454402
gk_positioning	gk_positioning	836	183142	0.454402
sprint_speed	sprint_speed	836	183142	0.454402
ball_control	ball_control	836	183142	0.454402
overall_rating	overall_rating	836	183142	0.454402
potential	potential	836	183142	0.454402
preferred_foot	preferred_foot	836	183142	0.454402
defensive_work_rate	defensive_work_rate	836	183142	0.454402
crossing	crossing	836	183142	0.454402
finishing	finishing	836	183142	0.454402
acceleration	acceleration	836	183142	0.454402
short_passing	short_passing	836	183142	0.454402

	column_name	No. Of Missing rows	No.of rows	percent_missing
heading_accuracy	heading_accuracy	836	183142	0.454402
dribbling	dribbling	836	183142	0.454402
free_kick_accuracy	free_kick_accuracy	836	183142	0.454402
long_passing	long_passing	836	183142	0.454402
gk_reflexes	gk_reflexes	836	183142	0.454402
curve	curve	2713	181265	1.474633
vision	vision	2713	181265	1.474633
jumping	jumping	2713	181265	1.474633
sliding_tackle	sliding_tackle	2713	181265	1.474633
balance	balance	2713	181265	1.474633
agility	agility	2713	181265	1.474633
volleys	volleys	2713	181265	1.474633
attacking_work_rate	attacking_work_rate	3230	180748	1.755645

In [44]:

```
# Replacing Null/NA values with Their mean for features having interger or float datatype
for col in df.select_dtypes(['int64','float64']):
    df[col]=df[col].fillna((df[col].mean()))
```

In [45]:

```
columns = df.columns
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': columns,'No. Of Missing rows':df.isnull().s
print("The Null/NA missing Data in Basket ball data is : \n")
missing_value_df.sort_values('percent_missing')
```

```
The Null/NA missing Data in Basket ball data is :
```

Out[45]:

	column_name	No. Of Missing rows	No.of rows	percent_missing
id	id	0	183978	0.000000
balance	balance	0	183978	0.000000
shot_power	shot_power	0	183978	0.000000
jumping	jumping	0	183978	0.000000
stamina	stamina	0	183978	0.000000
strength	strength	0	183978	0.000000
long_shots	long_shots	0	183978	0.000000
aggression	aggression	0	183978	0.000000
interceptions	interceptions	0	183978	0.000000
positioning	positioning	0	183978	0.000000
vision	vision	0	183978	0.000000
penalties	penalties	0	183978	0.000000
marking	marking	0	183978	0.000000
standing_tackle	standing_tackle	0	183978	0.000000
sliding_tackle	sliding_tackle	0	183978	0.000000
gk_diving	gk_diving	0	183978	0.000000
gk_handling	gk_handling	0	183978	0.000000
gk_kicking	gk_kicking	0	183978	0.000000
reactions	reactions	0	183978	0.000000
agility	agility	0	183978	0.000000
sprint_speed	sprint_speed	0	183978	0.000000
acceleration	acceleration	0	183978	0.000000
player_fifa_api_id	player_fifa_api_id	0	183978	0.000000
player_api_id	player_api_id	0	183978	0.000000
date	date	0	183978	0.000000
overall_rating	overall_rating	0	183978	0.000000
potential	potential	0	183978	0.000000
gk_positioning	gk_positioning	0	183978	0.000000
finishing	finishing	0	183978	0.000000

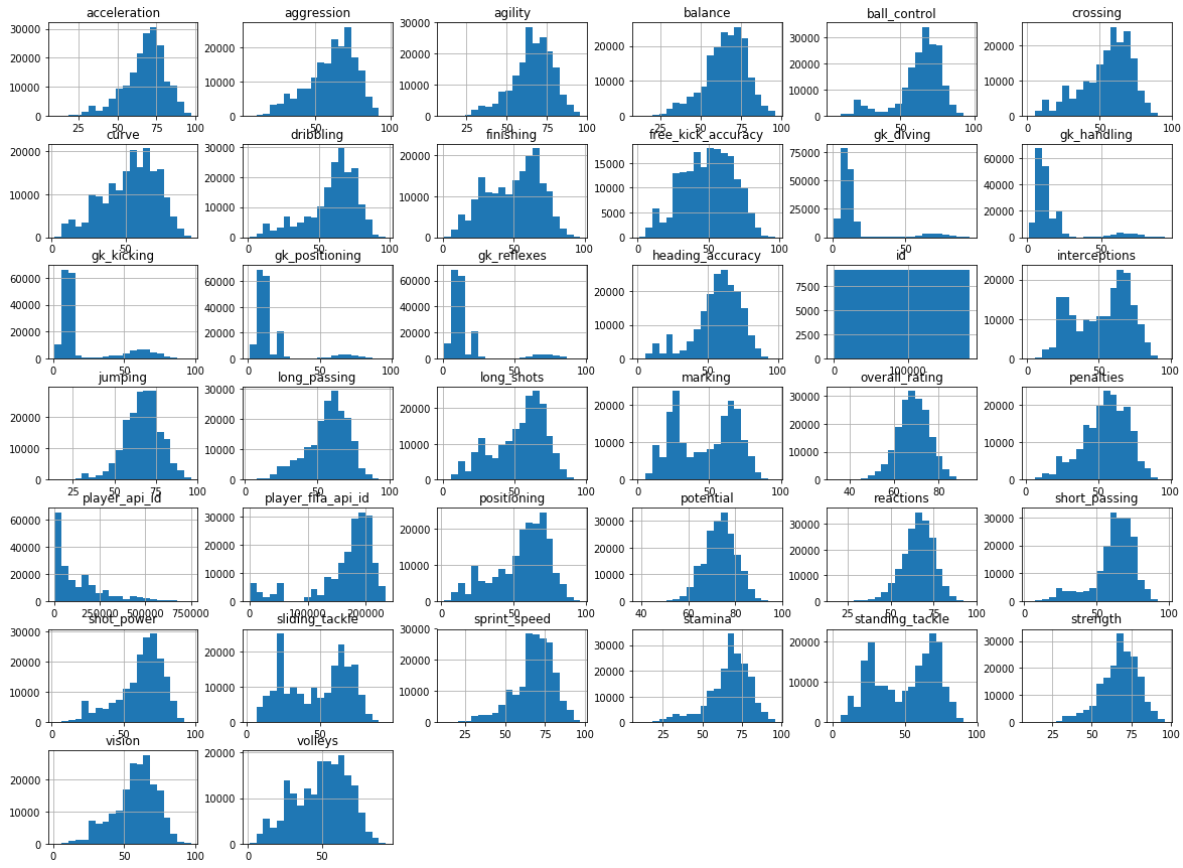
	column_name	No. Of Missing rows	No.of rows	percent_missing
crossing	crossing	0	183978	0.000000
short_passing	short_passing	0	183978	0.000000
volleys	volleys	0	183978	0.000000
dribbling	dribbling	0	183978	0.000000
curve	curve	0	183978	0.000000
free_kick_accuracy	free_kick_accuracy	0	183978	0.000000
long_passing	long_passing	0	183978	0.000000
ball_control	ball_control	0	183978	0.000000
heading_accuracy	heading_accuracy	0	183978	0.000000
gk_reflexes	gk_reflexes	0	183978	0.000000
defensive_work_rate	defensive_work_rate	836	183142	0.454402
preferred_foot	preferred_foot	836	183142	0.454402
attacking_work_rate	attacking_work_rate	3230	180748	1.755645

As stated , there is no null rows or missing data available iin Basketball data after missing row treatment except for Features those have object datatype.

Data Visualization

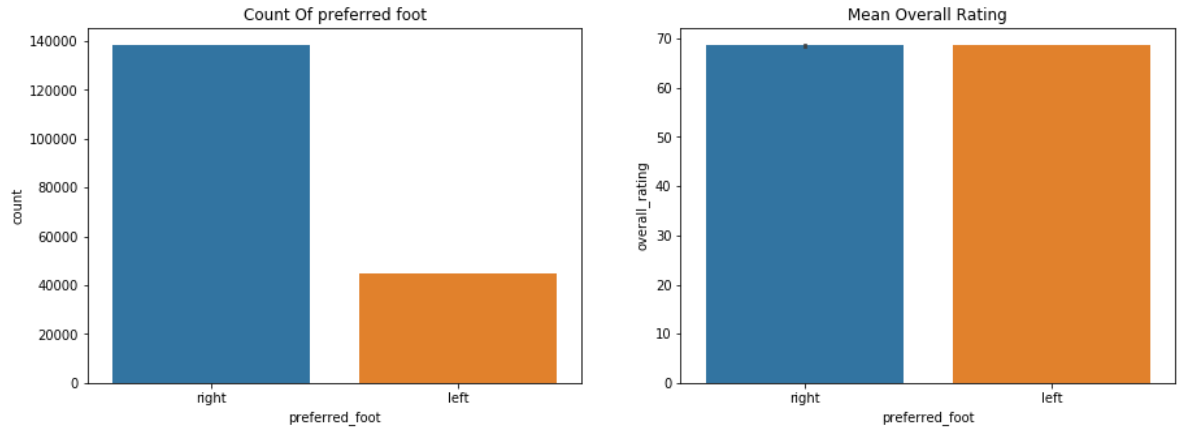
In [46]:

```
df.hist(bins=20, figsize=(20,15))
plt.show()
```



In [47]:

```
#Analyze the significane of Left and Right Foot.
plt.figure(figsize=(15, 5))
plt.subplot(1,2,1)
sns.countplot(df.preferred_foot)
plt.title('Count Of preferred foot')

plt.subplot(1,2,2)
sns.barplot(x='preferred_foot', y='overall_rating', data=df, estimator=np.mean)
plt.title('Mean Overall Rating')
plt.show()
```



Imputing target funtion :

In [48]:

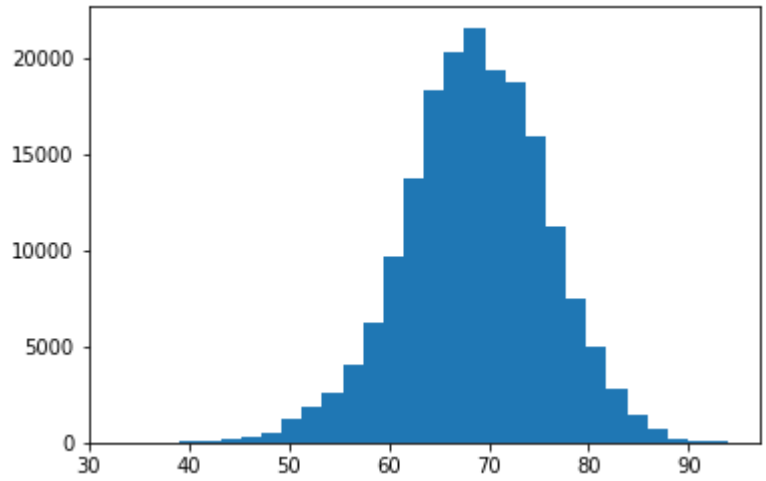
```
# Taking target variable out from the given dataset
target = df.pop("overall_rating")
target.describe()
```

Out[48]:

```
count    183978.000000
mean         68.600015
std           7.025124
min          33.000000
25%          64.000000
50%          69.000000
75%          73.000000
max          94.000000
Name: overall_rating, dtype: float64
```

In [49]:

```
plt.hist(target, 30, range=(33, 94))
plt.show()
```



almost normal distribution so we can impute mean value for missing value in target.

In [50]:

```
y = target.fillna(target.mean())
y.isnull().values.any()
```

Out[50]:

```
False
```

Data Exploration :

In [51]:

```
df.columns
```

Out[51]:

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

In [52]:

```
for col in df.columns:
    unique_cat = len(df[col].unique())
    print("{col}--> {unique_cat}..{typ}".format(col=col, unique_cat=unique_cat, typ=df[col]
```

```
id--> 183978..int64
player_fifa_api_id--> 11062..int64
player_api_id--> 11060..int64
date--> 197..object
potential--> 57..float64
preferred_foot--> 3..object
attacking_work_rate--> 9..object
defensive_work_rate--> 20..object
crossing--> 96..float64
finishing--> 98..float64
heading_accuracy--> 97..float64
short_passing--> 96..float64
volleys--> 94..float64
dribbling--> 98..float64
curve--> 93..float64
free_kick_accuracy--> 98..float64
long_passing--> 96..float64
ball_control--> 94..float64
acceleration--> 87..float64
sprint_speed--> 86..float64
agility--> 82..float64
reactions--> 79..float64
balance--> 82..float64
shot_power--> 97..float64
jumping--> 80..float64
stamina--> 85..float64
strength--> 83..float64
long_shots--> 97..float64
aggression--> 92..float64
interceptions--> 97..float64
positioning--> 96..float64
vision--> 98..float64
penalties--> 95..float64
marking--> 96..float64
standing_tackle--> 96..float64
sliding_tackle--> 95..float64
gk_diving--> 94..float64
gk_handling--> 91..float64
gk_kicking--> 98..float64
gk_positioning--> 95..float64
gk_reflexes--> 93..float64
```

we can see only four features have the type 'object'. here the feature named 'date' has no significance in this problem so can ignore it and perform one hot encoding on the rest of 3 features.

In [53]:

```
dummy_df = pd.get_dummies(df, columns=['preferred_foot', 'attacking_work_rate', 'defensive_
dummy_df.head()
```

Out[53]:

	id	player_fifa_api_id	player_api_id	date	potential	crossing	finishing	heading_accuracy
0	1	218353	505942	2016-02-18 00:00:00	71.0	49.0	44.0	71.(
1	2	218353	505942	2015-11-19 00:00:00	71.0	49.0	44.0	71.(
2	3	218353	505942	2015-09-21 00:00:00	66.0	49.0	44.0	71.(
3	4	218353	505942	2015-03-20 00:00:00	65.0	48.0	43.0	70.(
4	5	218353	505942	2007-02-22 00:00:00	65.0	48.0	43.0	70.(

5 rows × 67 columns

In [54]:

```
X = dummy_df.drop(['id', 'date'], axis=1)
```

Split Train and Test Dateset

In [55]:

```
# We are selecting 75:25 ration for train and test dataset
# shuffle paramter for shuffling the datasets before splitting
X_train, X_test , y_train , y_test = train_test_split(X , y , test_size=0.25 , random_state
```

In [57]:

```
# Details of Tarin and test dataset is :
print("The No. of rows in Features Training Dataset(X_train) is : {0}\n , No. of Columns in
    ".format(X_train.shape[0],X_train.shape[1]),"\n")
print("The No. of rows in Target Training Dataset(y_train) is : {0}".format(y_train.shape[0

print("The No. of rows in Features Test Dataset(X_Test) is : {0}\n , No. of Columns in Feat
    ".format(X_test.shape[0],X_test.shape[1]) , "\n")
print("The No. of rows in Target Test Dataset(y_test) is : {0}".format(y_test.shape[0]),"\r
```

```
The No. of rows in Features Training Dataset(X_train) is : 137983
 , No. of Columns in Features Training Dataset(X_train) is 65:

The No. of rows in Target Training Dataset(y_train) is : 137983

The No. of rows in Features Test Dataset(X_Test) is : 45995
 , No. of Columns in Features Training Dataset(X_Test) is 65:

The No. of rows in Target Test Dataset(y_test) is : 45995
```

Training different models

1. Linear Regression

In [61]:

```
# Apply linear regression model between Tagret and and independent features using Training
reg_model = LinearRegression()

# fitting of Target and Features
reg_model.fit(X_train, y=y_train)

# Calculation for R^2 (R sqaure)
reg_model_R_square =reg_model.score(X=X_train, y=y_train)

# Calculation for Predicted overall rating based upon features training dataset
y_pred_train = reg_model.predict(X=X_train)

print("The Coefficient of determination R^2 (R square) on Training Dataset : ",reg_model_R_
print("The Intercept Value for linear regression model is : " ,reg_model.intercept_ ,"\n")
print("The Predicted Overall rating based upon Features training dataset is (first 5 values
```

```
The Coefficient of determination R^2 (R square) on Training Dataset :  0.860
5299584793303

The Intercept Value for linear regression model is :  -0.46386415275662785

The Predicted Overall rating based upon Features training dataset is (first
5 values):
  [63.46803949 61.07223768 61.13335333 69.19083815 70.22149641]
```

In [62]:

```
# Mapping Coeffient values with their features name
df_reg_model_Coef = pd.DataFrame(list(zip(X_train.columns , reg_model.coef_)) ,columns=['Fe
print("The Features with their coeffient values(first 5 rows)  are:")
df_reg_model_Coef.head(5)
```

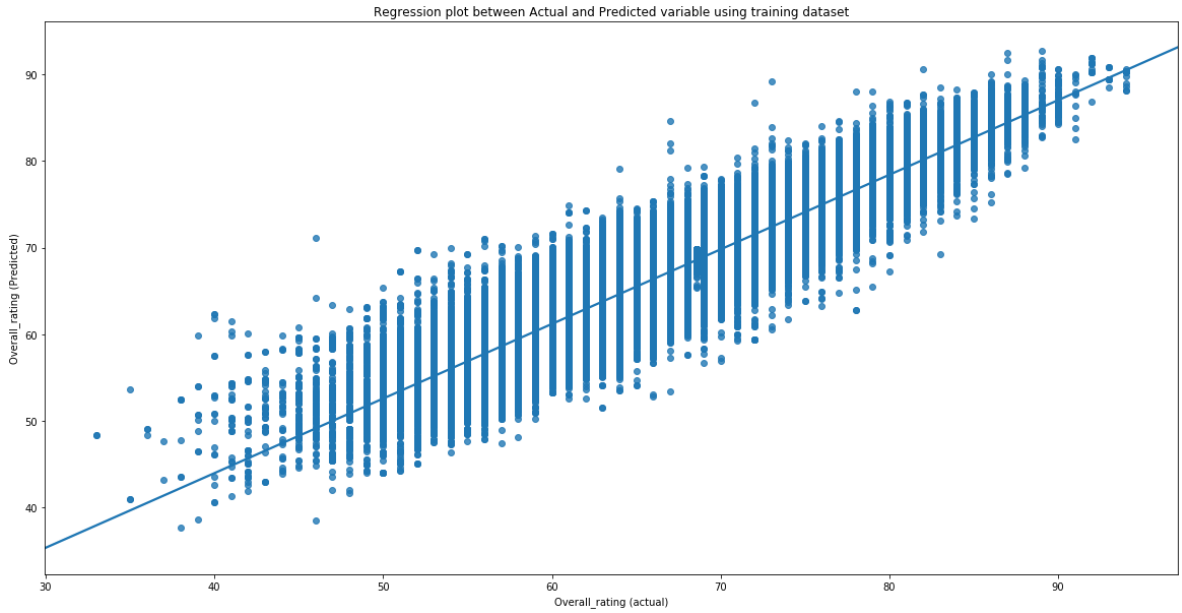
```
The Features with their coeffient values(first 5 rows)  are:
```

Out[62]:

	Features	Etimated coefficients
0	player_fifa_api_id	-0.000008
1	player_api_id	-0.000006
2	potential	0.445766
3	crossing	0.006179
4	finishing	0.014729

In [63]:

```
# Scatterplot between Actual Target values ( overall rating ) and Predicted target (predict
fig =plt.figure( figsize=(20,10))
sns.regplot(x=y_train, y=y_pred_train ,data=X_train) # searborn(sns).regplot for scatter pl
plt.xlabel("Overall_rating (actual)")
plt.ylabel("Overall_rating (Predicted)")
plt.title("Regression plot between Actual and Predicted variable using training dataset")
plt.show()
```



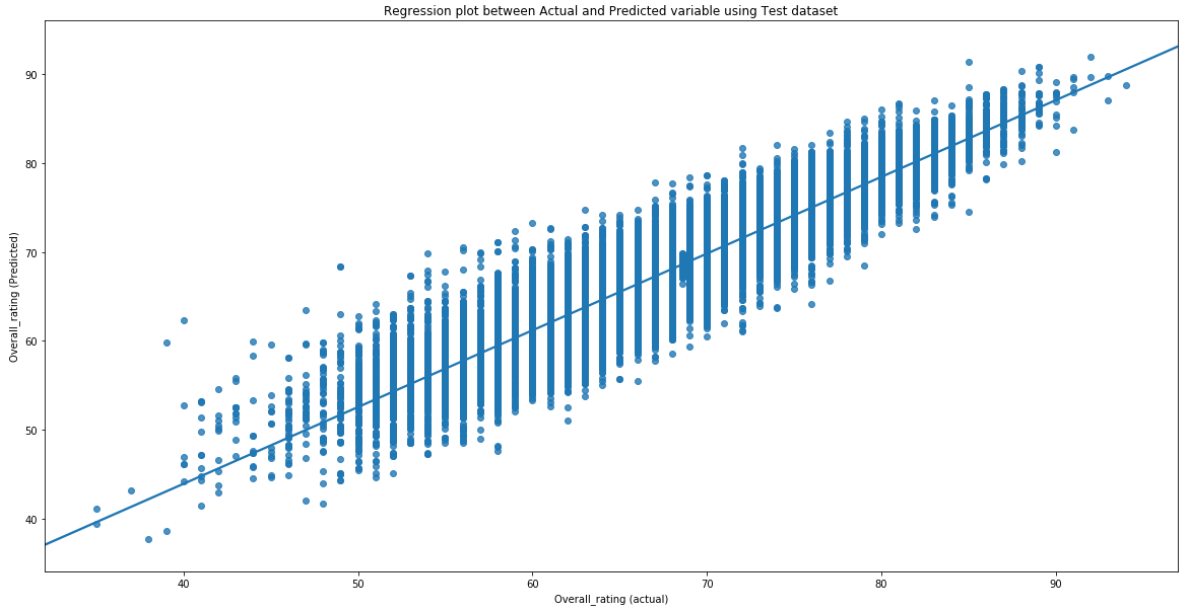
In [64]:

```
# Model Score predictions using Test Dataset
reg_model_R_square_Test= reg_model.score(X_test,y_test)
print("The Coefficient of determination R^2 (R square) on Test Dataset : ",reg_model_R_squa

# Prediction of overall rating based upon Test data set
y_pred_test= reg_model.predict(X_test)
```

```
The Coefficient of determination R^2 (R square) on Test Dataset :  0.8588100
435734953
```

In [65]:

```
# Scatterplot between Actual Target values ( overall rating ) and Predicted target (predict

fig =plt.figure( figsize=(20,10))
sns.regplot(x=y_test, y=y_pred_test ,data=X_test) # searborn(sns).regplot for scatter plot
plt.xlabel("Overall_rating (actual)")
plt.ylabel("Overall_rating (Predicted)")
plt.title("Regression plot between Actual and Predicted variable using Test dataset")
plt.show()
```



As we can see that , The line of goodness fit (regression line) is increasing upwards , for both training and test dataset, as the data has been plot between player's actual overall rating and player's predicted rating return by regression model.

Evaluate Model Performance

In [66]:

```
# Comaprison of Actual and Predicted Overall_rating on Test dataset
df_OverallRating_Test = pd.DataFrame({"Actual_rating":y_test,"Predicted_rating":reg_model.p
df_OverallRating_Test.head(10)
```

Out[66]:

	Actual_rating	Predicted_rating
171686	67.0	67.604273
145437	75.0	75.972117
74187	77.0	77.851191
139872	57.0	58.046885
98791	59.0	59.518246
129261	69.0	67.315557
14511	76.0	76.120651
18498	81.0	79.953429
76653	85.0	79.783916
33935	71.0	71.940507

In [67]:

```
# Regression model evaluation on Training dataset
mean_absolute_error = metrics.mean_absolute_error(y_train,y_pred_train)

mean_sqaured_error = metrics.mean_squared_error(y_train,y_pred_train)

root_mean_sqaured_error = math.sqrt(mean_sqaured_error)

print('Mean Absolute Error on Training dataset:', mean_absolute_error)
print('Mean Squared Error on Training dataset:', mean_sqaured_error)
print('Root Mean Squared Error on Training dataset:', root_mean_sqaured_error)
```

```
Mean Absolute Error on Training dataset: 1.9899757886544318
Mean Squared Error on Training dataset: 6.935315459178042
Root Mean Squared Error on Training dataset: 2.6334987106847123
```

In [68]:

```
# Regression model evaluation on Training dataset

mean_absolute_error_test = metrics.mean_absolute_error(y_test,y_pred_test)

mean_sqaured_error_test= metrics.mean_squared_error(y_test,y_pred_test)

root_mean_sqaured_error_test = math.sqrt(mean_sqaured_error_test)

print('Mean Absolute Error on Test dataset:', mean_absolute_error_test)
print('Mean Squared Error on Test dataset:', mean_sqaured_error_test)
print('Root Mean Squared Error on Test dataset:', root_mean_sqaured_error_test)
```

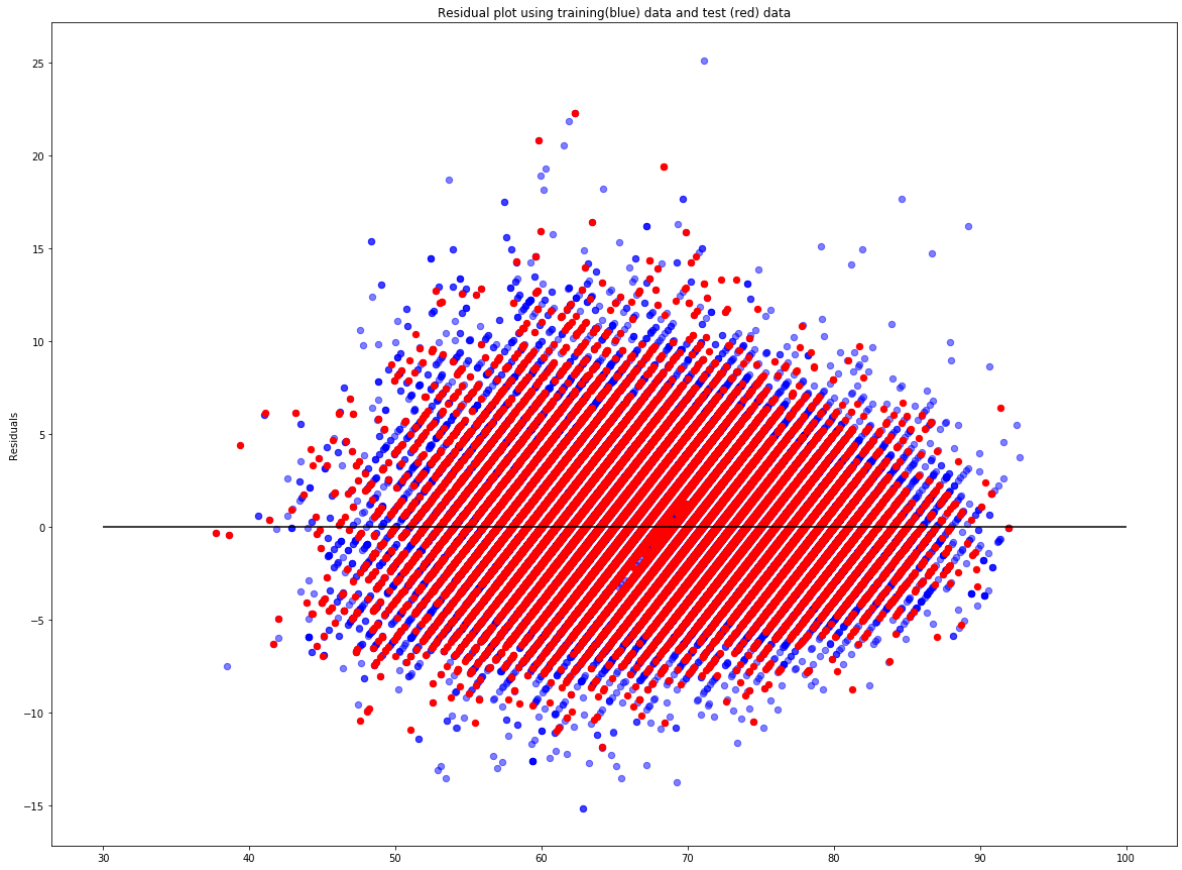
```
Mean Absolute Error on Test dataset: 1.9777630601817555
Mean Squared Error on Test dataset: 6.809442182822786
Root Mean Squared Error on Test dataset: 2.609490789947875
```

It can be observed that RMSE (Root Mean Squared Error) for Target variable in test set (predicted overall rating) is 2.6 which is less than that of training set. Hence, we can belive that our model performance is far good.

Residual Plot :Residual plots are way to visualize the errors in your data. If you have done a good job then your data should be randomly scattered around line zero.

In [69]:

```
fig= plt.figure(figsize=(20,15))
plt.scatter(reg_model.predict(X_train),reg_model.predict(X_train)-y_train , c='b',s=40, alp
plt.scatter(reg_model.predict(X_test),reg_model.predict(X_test)-y_test , c='r',s=40)
plt.hlines(y=0, xmin=30, xmax=100)
plt.title('Residual plot using training(blue) data and test (red) data')
plt.ylabel('Residuals')
plt.show()
```



As we can see that, Data for both training and test data randomly distributed around horizonal line , which is line zero.

Cross Validation Scores

In [70]:

```
# Perform 10-fold cross validation using
scores = cross_val_score(reg_model, X, y, cv=10, scoring='neg_mean_squared_error')

print("Cross-validated scores:", scores , "\n")
print("Mean Cross-validated score:", scores.mean(),"\n")
print("Standard deviation of scores:", scores.std())
```

```
Cross-validated scores: [-6.89408546 -6.84609517 -6.86942    -7.21091123 -6.
85603457 -7.34920306
 -6.79678538 -6.84460979 -6.95591541 -6.81182933]

Mean Cross-validated score: -6.943488940506922

Standard deviation of scores: 0.17605880940257193
```

Fitting Data Using StasModel

In [71]:

```
import statsmodels.api as Stats
# Fitting of data using OLS (ordinary least sqaure method)
reg_model_OLS = Stats.OLS(endog=y_train , exog=X_train , hasconst=True).fit()
# Prediction of Overall rating return by Regression model
y_pred_OLS=reg_model_OLS.predict(X_train)
```

In [72]:

```
print(reg_model_OLS.summary())
```

```
                            OLS Regression Results

==============================================================================
==
Dep. Variable:         overall_rating   R-squared:                       0.8
61
Model:                            OLS   Adj. R-squared:                  0.8
60
Method:                 Least Squares   F-statistic:                 1.467e+
04
Date:                Sun, 18 Nov 2018   Prob (F-statistic):               0.
00
Time:                        11:35:41   Log-Likelihood:            -3.2940e+
05
No. Observations:              137983   AIC:                          6.589e+
05
Df Residuals:                  137924   BIC:                          6.595e+
05
Df Model:                          58

Covariance Type:            nonrobust

==============================================================================
==================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
------------------
player_fifa_api_id         -8.416e-06   1.71e-07    -49.291      0.000   -8.
75e-06   -8.08e-06
player_api_id              -6.422e-06   7.07e-08    -90.901      0.000   -6.
56e-06   -6.28e-06
potential                      0.4447      0.002    271.263      0.000
0.441       0.448
crossing                       0.0061      0.001      6.603      0.000
0.004       0.008
finishing                      0.0146      0.001     14.725      0.000
0.013       0.017
heading_accuracy               0.0569      0.001     64.241      0.000
0.055       0.059
short_passing                  0.0546      0.001     36.741      0.000
0.052       0.057
volleys                       -0.0073      0.001     -8.225      0.000
-0.009      -0.006
dribbling                      0.0089      0.001      7.017      0.000
0.006       0.011
curve                          0.0105      0.001     12.094      0.000
0.009       0.012
free_kick_accuracy             0.0092      0.001     11.964      0.000
0.008       0.011
long_passing                   0.0103      0.001     10.137      0.000
0.008       0.012
ball_control                   0.1151      0.002     66.976      0.000
0.112       0.119
acceleration                   0.0114      0.001      7.630      0.000
0.008       0.014
sprint_speed                   0.0167      0.001     11.719      0.000
```

```
0.014       0.019
agility                        -0.0118      0.001    -10.767      0.000
-0.014      -0.010
reactions                       0.1753      0.001    146.325      0.000
0.173       0.178
balance                         0.0045      0.001      5.331      0.000
0.003       0.006
shot_power                      0.0121      0.001     12.757      0.000
0.010       0.014
jumping                         0.0111      0.001     13.958      0.000
0.010       0.013
stamina                        -0.0018      0.001     -1.963      0.050
-0.004   -2.31e-06
strength                        0.0499      0.001     56.300      0.000
0.048       0.052
long_shots                     -0.0092      0.001     -9.335      0.000
-0.011      -0.007
aggression                      0.0112      0.001     14.963      0.000
0.010       0.013
interceptions                   0.0028      0.001      3.490      0.000
0.001       0.004
positioning                    -0.0048      0.001     -5.651      0.000
-0.007      -0.003
vision                         -0.0179      0.001    -18.758      0.000
-0.020      -0.016
penalties                       0.0077      0.001      9.409      0.000
0.006       0.009
marking                         0.0270      0.001     22.315      0.000
0.025       0.029
standing_tackle                 0.0079      0.001      5.710      0.000
0.005       0.011
sliding_tackle                 -0.0196      0.001    -17.087      0.000
-0.022      -0.017
gk_diving                       0.1620      0.002    102.630      0.000
0.159       0.165
gk_handling                     0.0238      0.002     11.417      0.000
0.020       0.028
gk_kicking                     -0.0440      0.001    -65.246      0.000
-0.045      -0.043
gk_positioning                  0.0448      0.002     21.636      0.000
0.041       0.049
gk_reflexes                     0.0194      0.002      9.492      0.000
0.015       0.023
preferred_foot_left             0.1820      0.076      2.405      0.016
0.034       0.330
preferred_foot_right            0.1898      0.075      2.537      0.011
0.043       0.336
attacking_work_rate_None        0.3140      0.092      3.420      0.001
0.134       0.494
attacking_work_rate_high        0.5002      0.084      5.962      0.000
0.336       0.665
attacking_work_rate_le         -0.6099      0.141     -4.318      0.000
-0.887      -0.333
attacking_work_rate_low         1.3774      0.087     15.868      0.000
1.207       1.547
attacking_work_rate_medium      0.3622      0.083      4.386      0.000
0.200       0.524
attacking_work_rate_norm       -0.0534      0.081     -0.660      0.509
-0.212       0.105
attacking_work_rate_stoc       -0.7224      0.158     -4.584      0.000
-1.031      -0.414
```

```
attacking_work_rate_y         -0.2947      0.141     -2.091      0.037
-0.571      -0.018
defensive_work_rate_0          0.4183      0.214      1.956      0.050
-0.001       0.837
defensive_work_rate_1          0.9500      0.148      6.416      0.000
0.660       1.240
defensive_work_rate_2          0.1401      0.163      0.862      0.389
-0.179       0.459
defensive_work_rate_3          0.4067      0.190      2.140      0.032
0.034       0.779
defensive_work_rate_4         -0.5258      0.259     -2.033      0.042
-1.033      -0.019
defensive_work_rate_5         -0.3233      0.191     -1.689      0.091
-0.698       0.052
defensive_work_rate_6         -0.3292      0.213     -1.548      0.122
-0.746       0.088
defensive_work_rate_7          0.5509      0.200      2.749      0.006
0.158       0.944
defensive_work_rate_8          0.8828      0.331      2.670      0.008
0.235       1.531
defensive_work_rate_9          0.4699      0.235      1.999      0.046
0.009       0.931
defensive_work_rate__0        -0.5015      0.075     -6.691      0.000
-0.648      -0.355
defensive_work_rate_ean       -0.6099      0.141     -4.318      0.000
-0.887      -0.333
defensive_work_rate_es        -0.2947      0.141     -2.091      0.037
-0.571      -0.018
defensive_work_rate_high       0.1602      0.110      1.459      0.145
-0.055       0.376
defensive_work_rate_low        0.3595      0.111      3.244      0.001
0.142       0.577
defensive_work_rate_medium    -0.0854      0.109     -0.785      0.432
-0.299       0.128
defensive_work_rate_o         -0.5209      0.103     -5.067      0.000
-0.722      -0.319
defensive_work_rate_ormal     -0.0534      0.081     -0.660      0.509
-0.212       0.105
defensive_work_rate_tocky     -0.7224      0.158     -4.584      0.000
-1.031      -0.414
==============================================================================
==
Omnibus:                    10988.497   Durbin-Watson:                   2.0
02
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            28341.7
89
Skew:                          -0.468   Prob(JB):                         0.
00
Kurtosis:                       5.013   Cond. No.                     1.42e+
16
==============================================================================
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The smallest eigenvalue is 4.18e-17. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [73]:

```
print("The Coefficient of determination R^2 (R square) on Training Dataset(OLS model) : ",r

print("The Predicted Overall rating based upon Features training dataset is (first 5 values
y_pred_OLS[0:5]
```

```
The Coefficient of determination R^2 (R square) on Training Dataset(OLS mode
l) :  0.8605203776501418

The Predicted Overall rating based upon Features training dataset is (first
5 values) (OLS model):
```

Out[73]:

```
133965    63.475366
17293     61.085508
105272    61.149205
50368     69.186129
22911     70.214966
dtype: float64
```

As, We can see that when model is fitted with OLS method from Statsmodel, again it's Coefficient of determination is returning as 86%

2. Decision Tree

In [74]:

```
# Calling of DecisionTreeReegressor model
decision_tree_model = DecisionTreeRegressor()
#Fitting of Data into model (training data)
decision_tree_model.fit(X=X_train , y=y_train)
```

Out[74]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, 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 [75]:

```
# Prediction of overall rating
y_pred_dec_tree = decision_tree_model.predict(X_train)
print("Predicted overall ratings : (first 5 rows)")
y_pred_dec_tree[0:5]
```

```
Predicted overall ratings : (first 5 rows)
```

Out[75]:

```
array([64.33333333, 58.        , 65.        , 71.        , 68.        ])
```

In [76]:

```
# Accuracy of Decision tree model
dec_tree_Accuracy_score =decision_tree_model.score(X=X_train , y=y_train)
print("Accuracy of Decision Tree model to Predict Rating of players")
dec_tree_Accuracy_score*100
```

```
Accuracy of Decision Tree model to Predict Rating of players
```

Out[76]:

```
99.94208782299371
```

Conclusion:

As we observe, Decision tree model has predicted with accuracy (99%) and the Linear regression model has predicted accuracy with 84%. Looks like, Decision tree performs well for the given task. i.e Predicting players rating.