Show Code

1. Importing Necessary Packages

```
All the required packages have been installed
```

2. Reading the necessary files

```
The necessary file has been read.
```

3. Analysing the dataset

3.1. Displaying the first 10 tuples

Out[3]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	h
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	h
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	h
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	h
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	
5	5	183277	E. Hazard	27	https://cdn.sofifa.org/players/4/19/183277.png	Belgium	
6	6	177003	L. Modrić	32	https://cdn.sofifa.org/players/4/19/177003.png	Croatia	h
7	7	176580	L. Suárez	31	https://cdn.sofifa.org/players/4/19/176580.png	Uruguay	h
8	8	155862	Sergio Ramos	32	https://cdn.sofifa.org/players/4/19/155862.png	Spain	h
9	9	200389	J. Oblak	25	https://cdn.sofifa.org/players/4/19/200389.png	Slovenia	h

10 rows × 89 columns

3.2.Exploring the columns of the dataset

Out[4]:

```
Index(['Unnamed: 0', 'ID', 'Name', 'Age', 'Photo', 'Nationality', 'Flag',
       'Overall', 'Potential', 'Club', 'Club Logo', 'Value', 'Wage', 'Spec
ial',
       'Preferred Foot', 'International Reputation', 'Weak Foot',
       'Skill Moves', 'Work Rate', 'Body Type', 'Real Face', 'Position',
       'Jersey Number', 'Joined', 'Loaned From', 'Contract Valid Until',
       'Height', 'Weight', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW',
       'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM',
       'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'Crossing',
       'Finishing', 'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribbli
ng',
       'Curve', 'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleratio
n',
       'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
       'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
       'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
       'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandli
ng',
       'GKKicking', 'GKPositioning', 'GKReflexes', 'Release Clause'],
      dtype='object')
```

3.3 Checking the present dimension of the data set

```
No. of Rows in the data:  18207
No. of Columns in the data:  89
```

3.4 Checking number of null values present in each column of the dataset

```
Data contains NULL values: True
```

3.5. Checking for duplicated values

```
Number of DUPLICATED values in the Data: 0
```

4. Data Visualization and Preprocessing

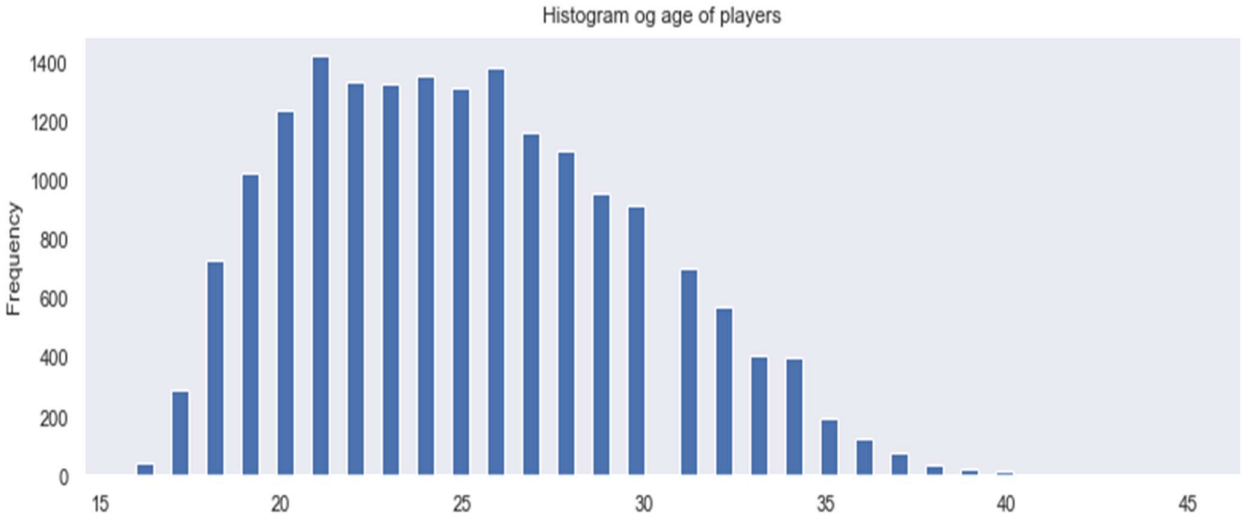
On proper analyzing the data set it is found that some rows contains more than 70% of null entries so such rows are removed

```
Such rows are deleted using drop
```

4.1 Plotting age and its frequency

Out[9]:

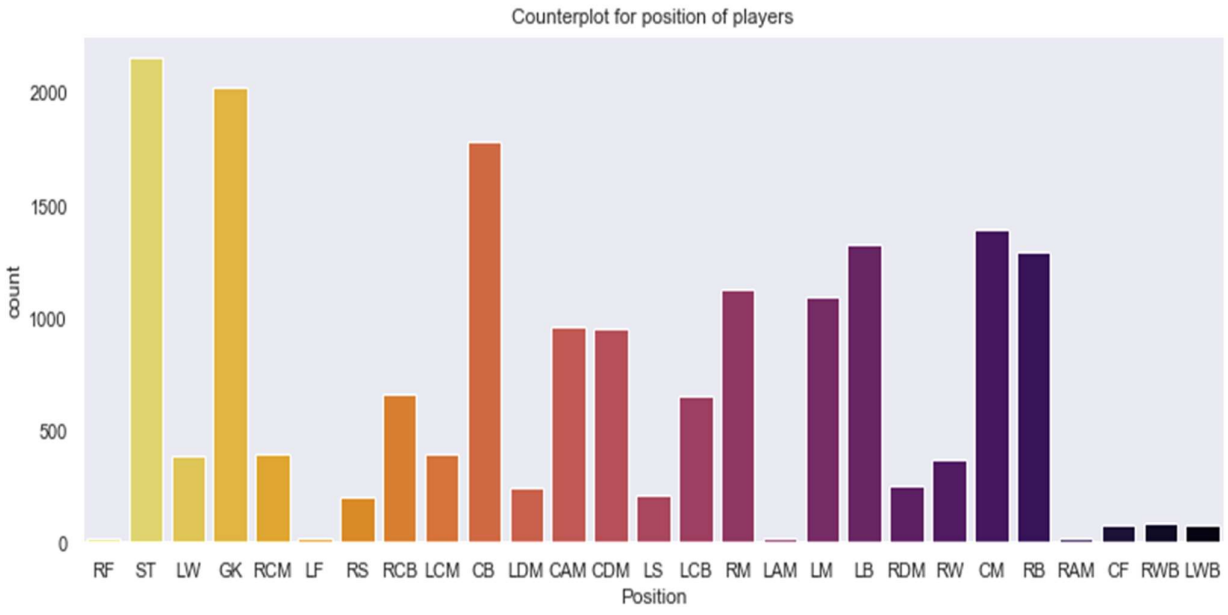
```
Text(0.5, 1.0, 'Histogram of age of players')
```



4.2 Counterplot for position of players

Out[10]:

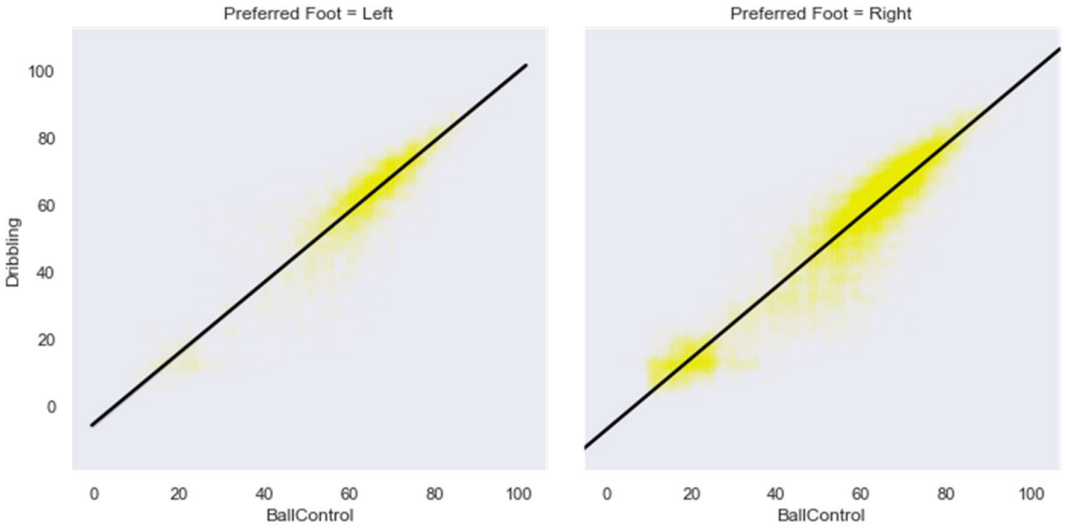
```
Text(0.5, 1.0, 'Counterplot for position of players')
```



4.3 Comparison b/w left and right footer players

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x194840d7278>
```



4.4 Plotting nationality v/s frequency of players in that nation

Out[12]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 <a list of 11 Text xticklabel objects>)
```

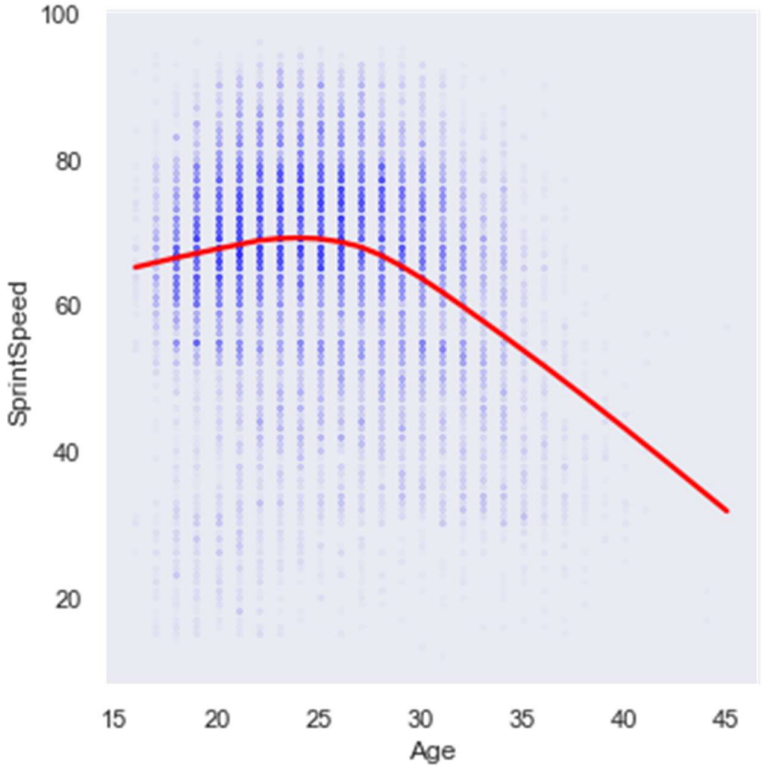
4.5 Dropping unneccesary and unwanted columns from the data set

```
Columns such as 'Unnamed: 0', 'ID', 'Name', 'Photo', 'Nationality', 'Fla
g','Club', 'Club Logo','Wage','Preferred Foot','Weak Foot','Real Face','Je
rsey Number', 'Joined', 'Loaned From', 'Contract Valid Until','Release Cla
use' are deleted.
```

4.6 Visualizing age vs sprintspeed

Out[14]:

```
<seaborn.axisgrid.FacetGrid at 0x19484302a90>
```



The above graph shows us that as the age increases the sprintspeed decreases

4.7 Checking for null values

```
Data contains NULL values: True
```

4.7.1 The above columns that have the null values have numeric values stored in string, before imputing we should process such rows

Out[16]:

	LS	ST	RS	LW	LF	CF	RF	RW	LAM	CAM	...	LWB	LDM	CDM	R
0	88+2	88+2	88+2	92+2	93+2	93+2	93+2	92+2	93+2	93+2	...	64+2	61+2	61+2	61
1	91+3	91+3	91+3	89+3	90+3	90+3	90+3	89+3	88+3	88+3	...	65+3	61+3	61+3	61
2	84+3	84+3	84+3	89+3	89+3	89+3	89+3	89+3	89+3	89+3	...	65+3	60+3	60+3	60
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
4	82+3	82+3	82+3	87+3	87+3	87+3	87+3	87+3	88+3	88+3	...	77+3	77+3	77+3	77
5	83+3	83+3	83+3	89+3	88+3	88+3	88+3	89+3	89+3	89+3	...	66+3	63+3	63+3	63
6	77+3	77+3	77+3	85+3	84+3	84+3	84+3	85+3	87+3	87+3	...	82+3	81+3	81+3	81
7	87+5	87+5	87+5	86+5	87+5	87+5	87+5	86+5	85+5	85+5	...	69+5	68+5	68+5	68
8	73+3	73+3	73+3	70+3	71+3	71+3	71+3	70+3	71+3	71+3	...	81+3	84+3	84+3	84
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N

10 rows × 26 columns

4.7.1.1 The columns such as LS, ST and so on contains strings as 90+2 so for converting these strings to float a function string_to_number is defined below

```
The function string_to_number is defined to convert String type number to
Integer type number
```

4.7.1.2 Applying the function to all the required columns

```
string_to_number function is applied to all columns between index 12 to 38
successfully
```

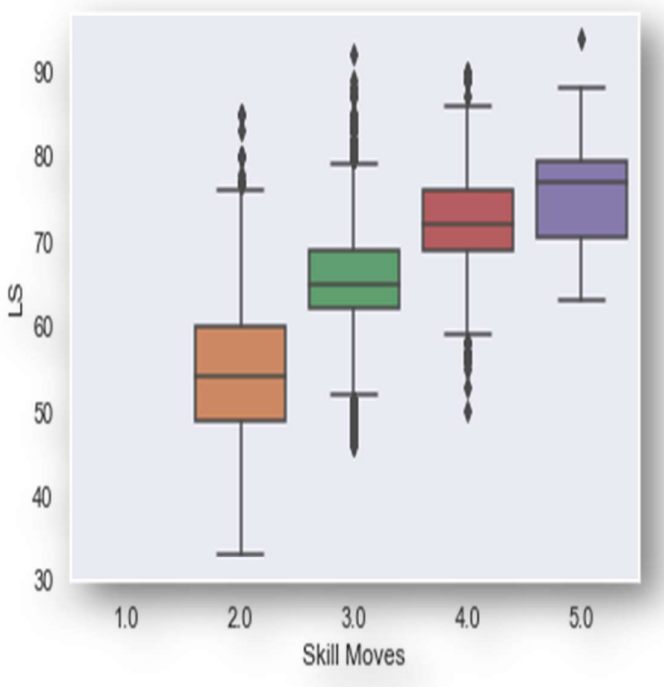
4.7.1.3 Displaying the processed columns

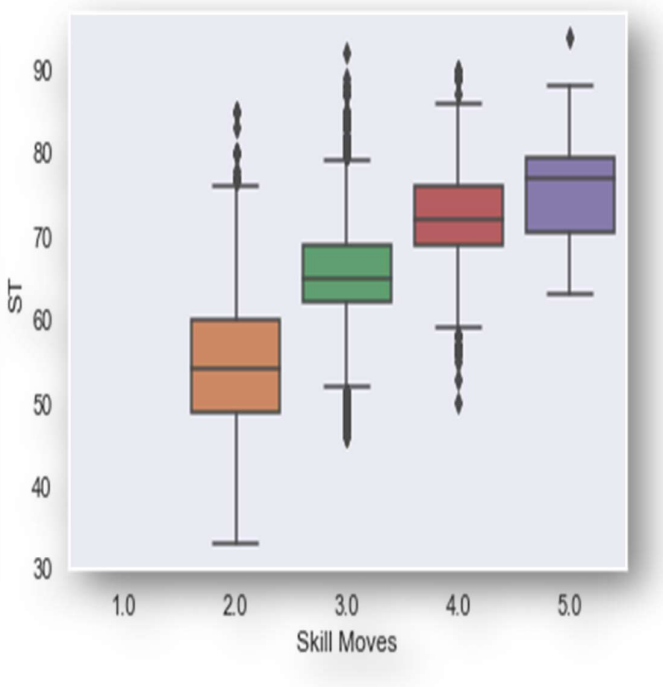
Out[19]:

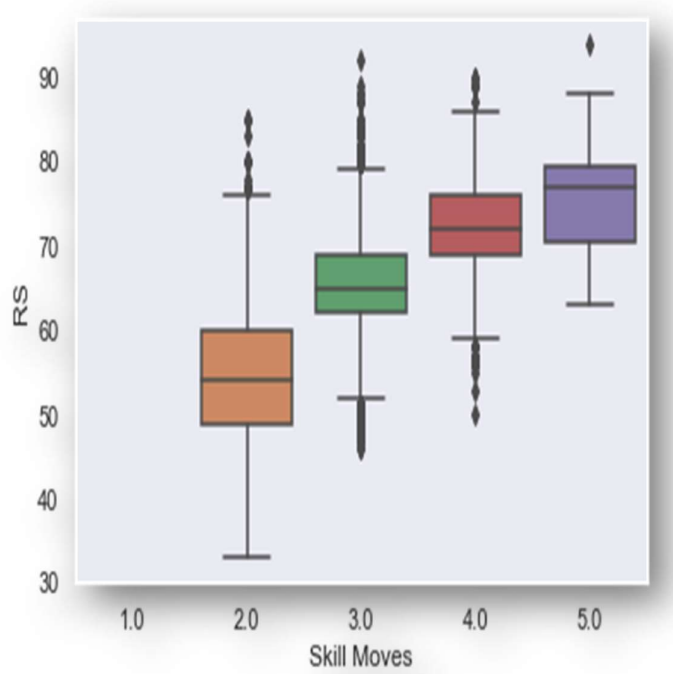
	LS	ST	RS	LW	LF	CF	RF	RW	LAM	CAM	...	LWB	LDM	CDM	RDM
0	90.0	90.0	90.0	94.0	95.0	95.0	95.0	94.0	95.0	95.0	...	66.0	63.0	63.0	63.0
1	94.0	94.0	94.0	92.0	93.0	93.0	93.0	92.0	91.0	91.0	...	68.0	64.0	64.0	64.0
2	87.0	87.0	87.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0	...	68.0	63.0	63.0	63.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	85.0	85.0	85.0	90.0	90.0	90.0	90.0	90.0	91.0	91.0	...	80.0	80.0	80.0	80.0
5	86.0	86.0	86.0	92.0	91.0	91.0	91.0	92.0	92.0	92.0	...	69.0	66.0	66.0	66.0
6	80.0	80.0	80.0	88.0	87.0	87.0	87.0	88.0	90.0	90.0	...	85.0	84.0	84.0	84.0
7	92.0	92.0	92.0	91.0	92.0	92.0	92.0	91.0	90.0	90.0	...	74.0	73.0	73.0	73.0
8	76.0	76.0	76.0	73.0	74.0	74.0	74.0	73.0	74.0	74.0	...	84.0	87.0	87.0	87.0
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
10	90.0	90.0	90.0	86.0	89.0	89.0	89.0	86.0	86.0	86.0	...	64.0	65.0	65.0	65.0
11	81.0	81.0	81.0	84.0	85.0	85.0	85.0	84.0	87.0	87.0	...	82.0	85.0	85.0	85.0
12	67.0	67.0	67.0	64.0	65.0	65.0	65.0	64.0	65.0	65.0	...	79.0	84.0	84.0	84.0
13	80.0	80.0	80.0	88.0	87.0	87.0	87.0	88.0	90.0	90.0	...	72.0	73.0	73.0	73.0
14	75.0	75.0	75.0	80.0	80.0	80.0	80.0	80.0	82.0	82.0	...	88.0	90.0	90.0	90.0

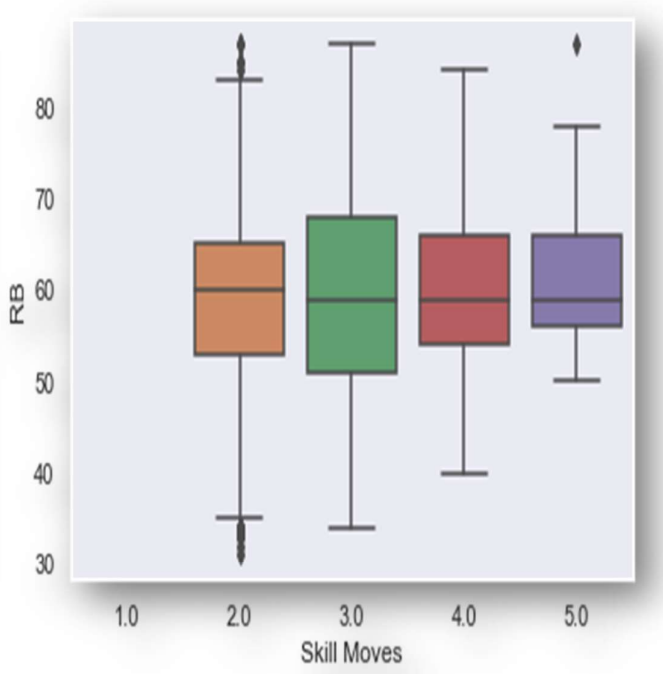
15 rows × 26 columns

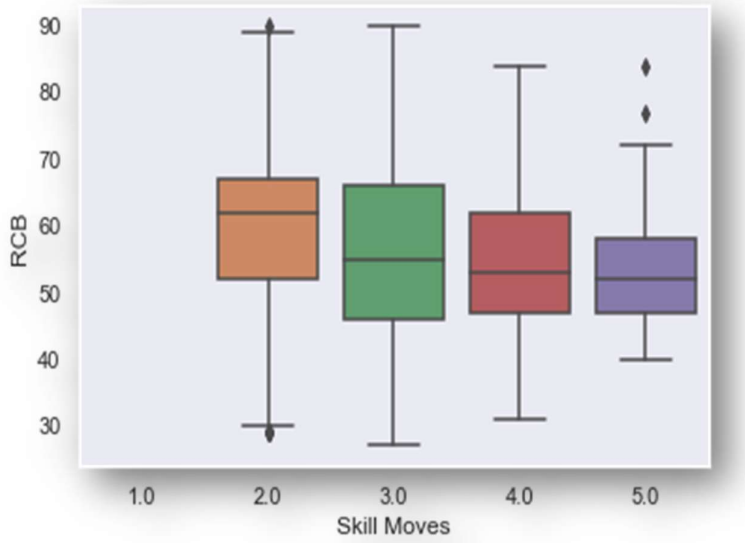
4.7.2 Plotting graphs on basis of "Skill Moves" for each row having null values











From the boxplot it is evident that NaN values occurs only for those players whose "Skill Values" is 1. Also analyzing the data set it is evident that players with Position as Goalkeeper have skill value equal to 1

4.7.2.1 Function to impute NAN values

```
The function imputeNan is defined to impute NaN values based on Skill Move
s
```

4.7.2.2 Appling the function to impute NAN values

```
imputeNan function is applied to Value column successfully
```

4.7.2.3 Displaying the processed columns

Out[23]:

	LS	ST	RS	LW	LF	CF	RF	RW	LAM	CAM	...	LWB	LDM	CDM	RDM	R
0	90.0	90.0	90.0	94.0	95.0	95.0	95.0	94.0	95.0	95.0	...	66.0	63.0	63.0	63.0	6
1	94.0	94.0	94.0	92.0	93.0	93.0	93.0	92.0	91.0	91.0	...	68.0	64.0	64.0	64.0	6
2	87.0	87.0	87.0	92.0	92.0	92.0	92.0	92.0	92.0	92.0	...	68.0	63.0	63.0	63.0	6
3	14.0	7.0	8.0	5.0	10.0	7.0	6.0	12.0	6.0	7.0	...	8.0	8.0	6.0	11.0	1
4	85.0	85.0	85.0	90.0	90.0	90.0	90.0	90.0	91.0	91.0	...	80.0	80.0	80.0	80.0	8
5	86.0	86.0	86.0	92.0	91.0	91.0	91.0	92.0	92.0	92.0	...	69.0	66.0	66.0	66.0	6
6	80.0	80.0	80.0	88.0	87.0	87.0	87.0	88.0	90.0	90.0	...	85.0	84.0	84.0	84.0	8
7	92.0	92.0	92.0	91.0	92.0	92.0	92.0	91.0	90.0	90.0	...	74.0	73.0	73.0	73.0	7
8	76.0	76.0	76.0	73.0	74.0	74.0	74.0	73.0	74.0	74.0	...	84.0	87.0	87.0	87.0	8
9	9.0	9.0	11.0	13.0	10.0	5.0	14.0	10.0	14.0	7.0	...	5.0	11.0	14.0	12.0	1

10 rows × 26 columns

Finally, all the preprocessing for null values have been done

4.8 Processing Categorial values

4.8.1.1 Value of players are given in euros as string, "torupees" converts the euros to rupees

```
The function torupees is defined to convert Euros to Rupees
```

4.8.1.2 Applying "torupees" function to Value column

```
torupees function is applied to Value column successfully
```

4.8.1.3 Displaying the changed Value

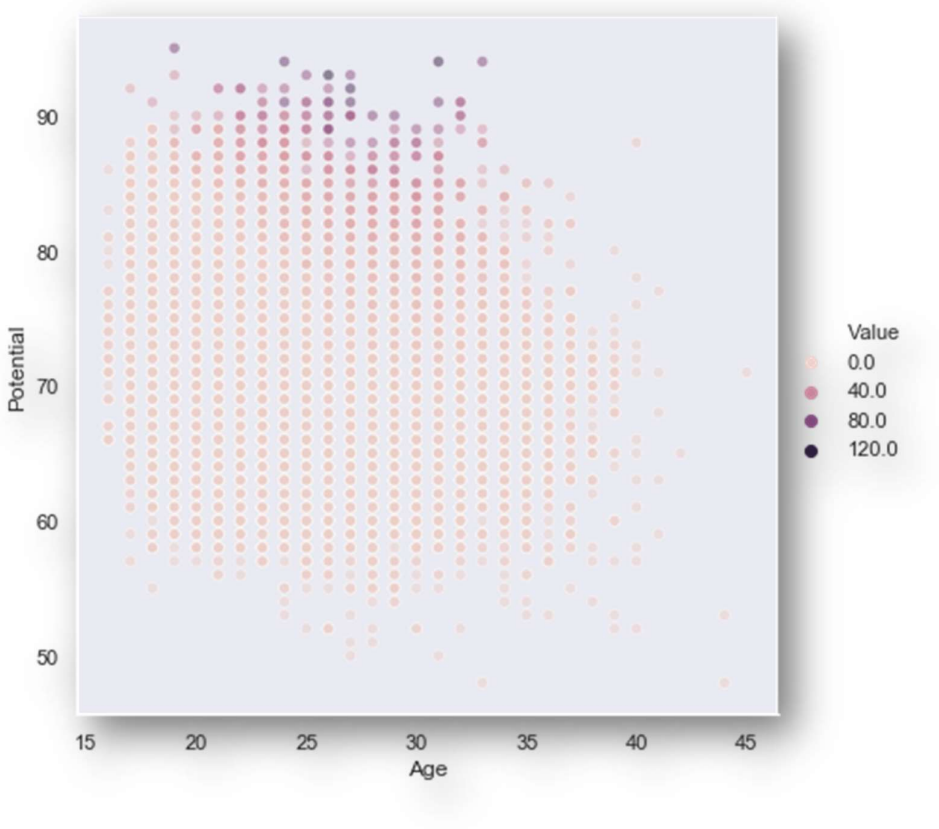
Out[26]:

	Value
0	110500000.0
1	77000000.0
2	118500000.0
3	72000000.0
4	102000000.0

4.8.2 Comparison b/w age and potential

Out[27]:

```
<seaborn.axisgrid.FacetGrid at 0x194832e3da0>
```



4.8.3.1 Checking the unique values present in Column Body Type

Out[28]:

```
array(['Messi', 'C. Ronaldo', 'Neymar', 'Lean', 'Normal', 'Courtois',
       'Stocky', 'PLAYER_BODY_TYPE_25', 'Shaqiri', 'Akinfenwa'],
      dtype=object)
```

4.8.3.2 Changing the values to a proper type

```
All the body types are converted in three groups
```

4.8.3.3 Displaying the body types

```
The body types: ['Lean' 'Normal' 'Stocky']
```

4.8.4 The column Height contains values in inches and feet,heightcoversion is used for converting the height in inch

4.8.4.1 Displaying the sample height.

```
The samples of the height
```

Out[31]:

```
0      5'7
1      6'2
2      5'9
3      6'4
4     5'11
5      5'8
6      5'8
7      6'0
8      6'0
9      6'2
10     6'0
11     6'0
12     6'2
13     5'8
14     5'6
15    5'10
16     6'2
17     5'9
18     6'2
19     6'6
Name: Height, dtype: object
```

4.8.4.2 Defining a function to convert height in inches only

```
The 'heightconversion' function is defined
```

4.8.4.3 Applying "heightconversion" function to the Height column

Out[36]:

```
0    67
1    74
2    69
3    76
4    71
Name: Height, dtype: int64
```

4.8.5 Weight column contains "lbs" as unit which needs to be removed,weight function is defined for this.

Out[37]:

```
0    159lbs
1    183lbs
2    150lbs
3    168lbs
4    154lbs
Name: Weight, dtype: object
```

4.8.5.1 Defining a function to convert weight in floating point.

4.8.5.2 Applying "weight" function to the Weight column

Out[40]:

```
0    159.0
1    183.0
2    150.0
3    168.0
4    154.0
Name: Weight, dtype: float64
```

4.8.6 Columns such as Position, Work Rate and Body Type contains categorial values which needs to be removed using Label Encoder.

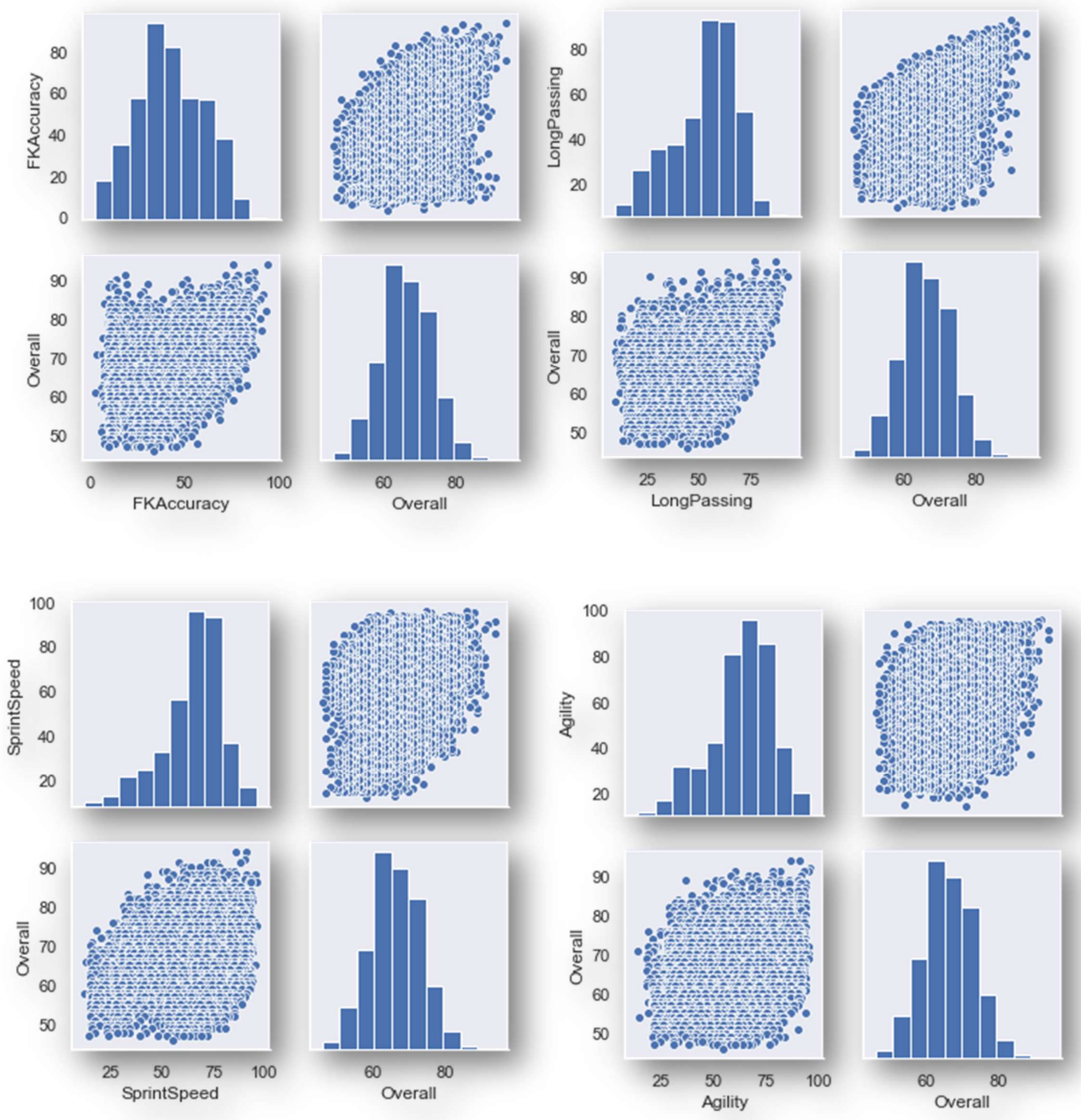
```
Label Encoder used and the changed values sample is printed.
   Position  Work Rate  Body Type
0        21          8          0
1        26          1          1
2        14          2          0
3         5          8          0
4        19          0          1
5        11          2          1
```

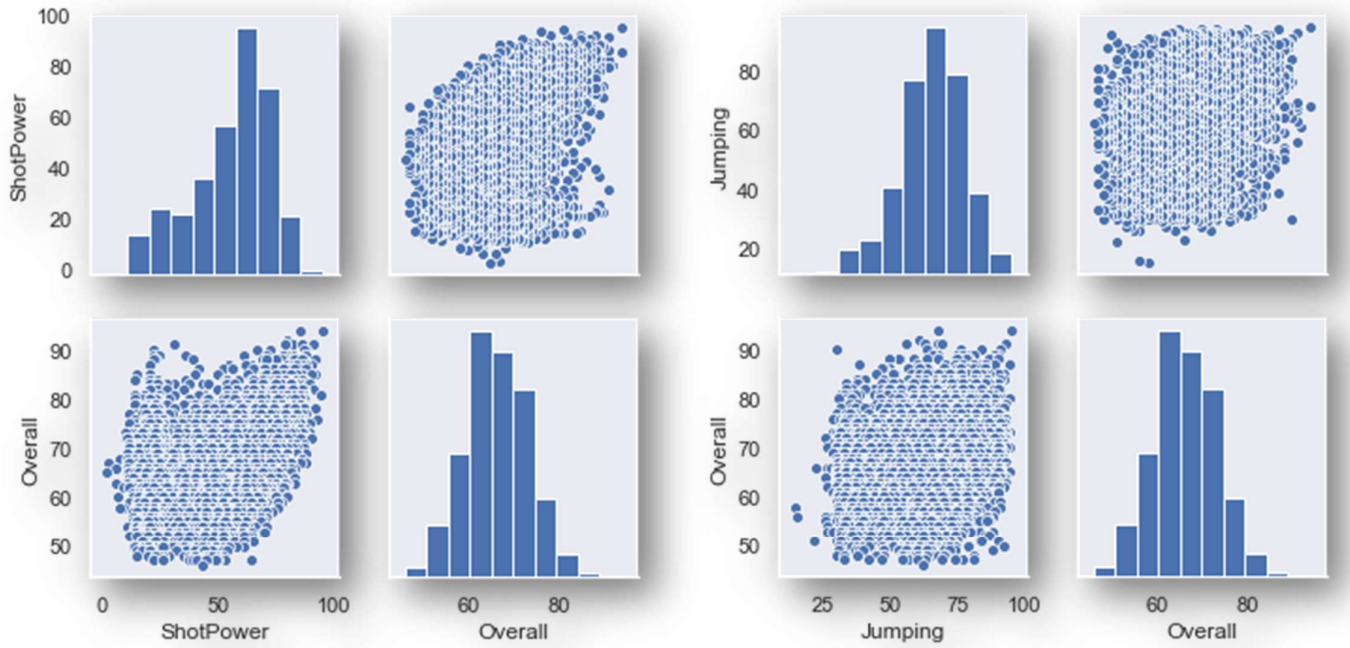
Shape of the dataset

Out[43]:

```
(18147, 72)
```

4.9 Checking for Linear Features and Normalized features using Pairplot





4.9.1 PairPlot shows that most of the features does not have a linear relation with the dependent target.

```
Dropping all the Non-Linear Features
```

4.9.2 Columns such as 'Work Rate', 'Body Type', 'Position' are not normalized and log transformation also does not help.

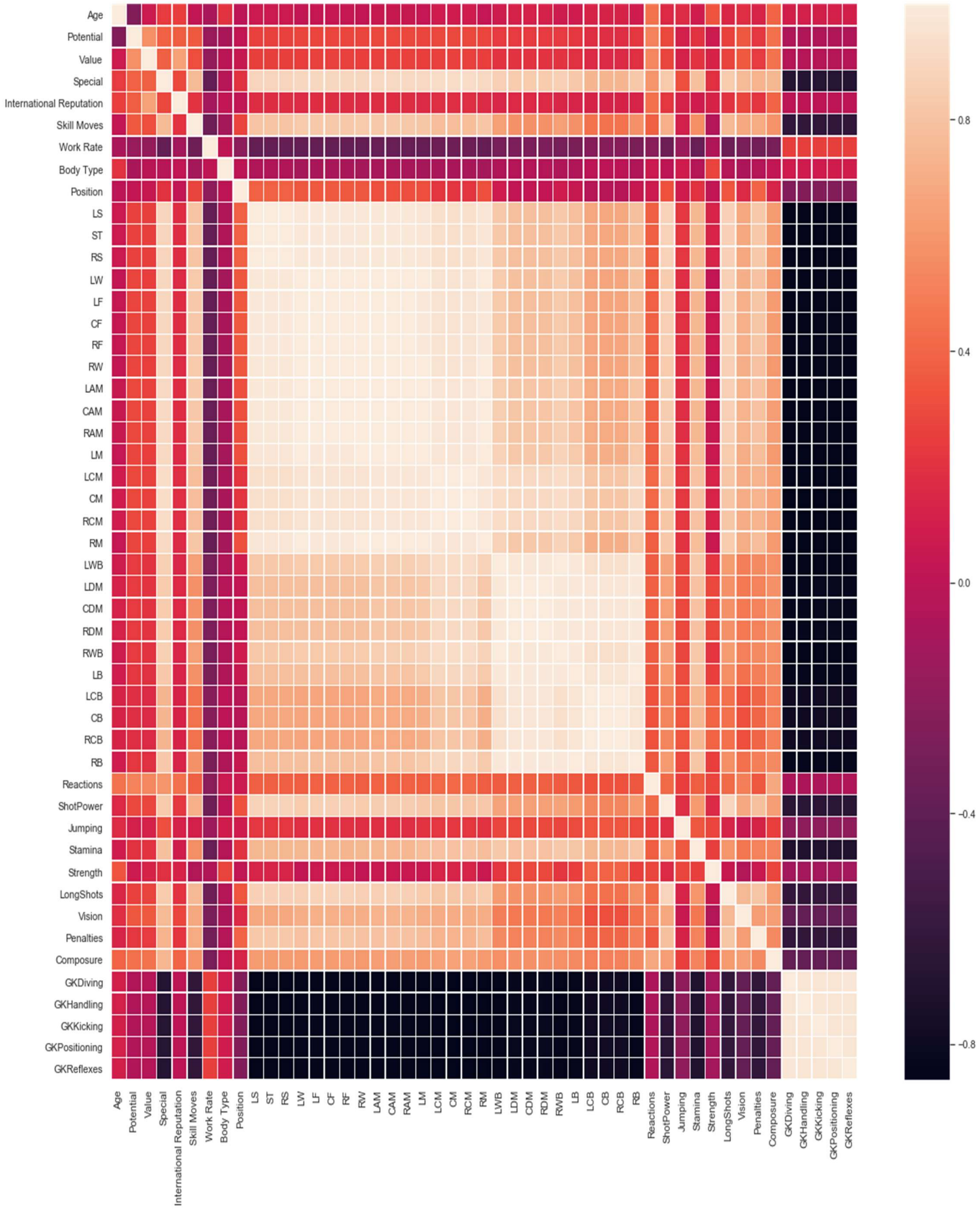
```
Normalizing Work Rate, Body Type, Position columns does not help as data
does not get normalized so they are not normalized.
```

The data contains the column that is to be predicted. This column is stored in "y" and dropped from "data" so that Pearson Correlation Coefficient can be found.

```
Storing the target or dependent variable
```

4.10 Visualizing the Correlation Matrix using HeatMap

Out[48]:<matplotlib.axes._subplots.AxesSubplot at 0x194851b42b0>



4.10.1 Checking for correlation coefficient for different columns for which value is greater or equal to 0.9 using function

```
Columns having coefficient greater than 0.9 needs to be deleted
```

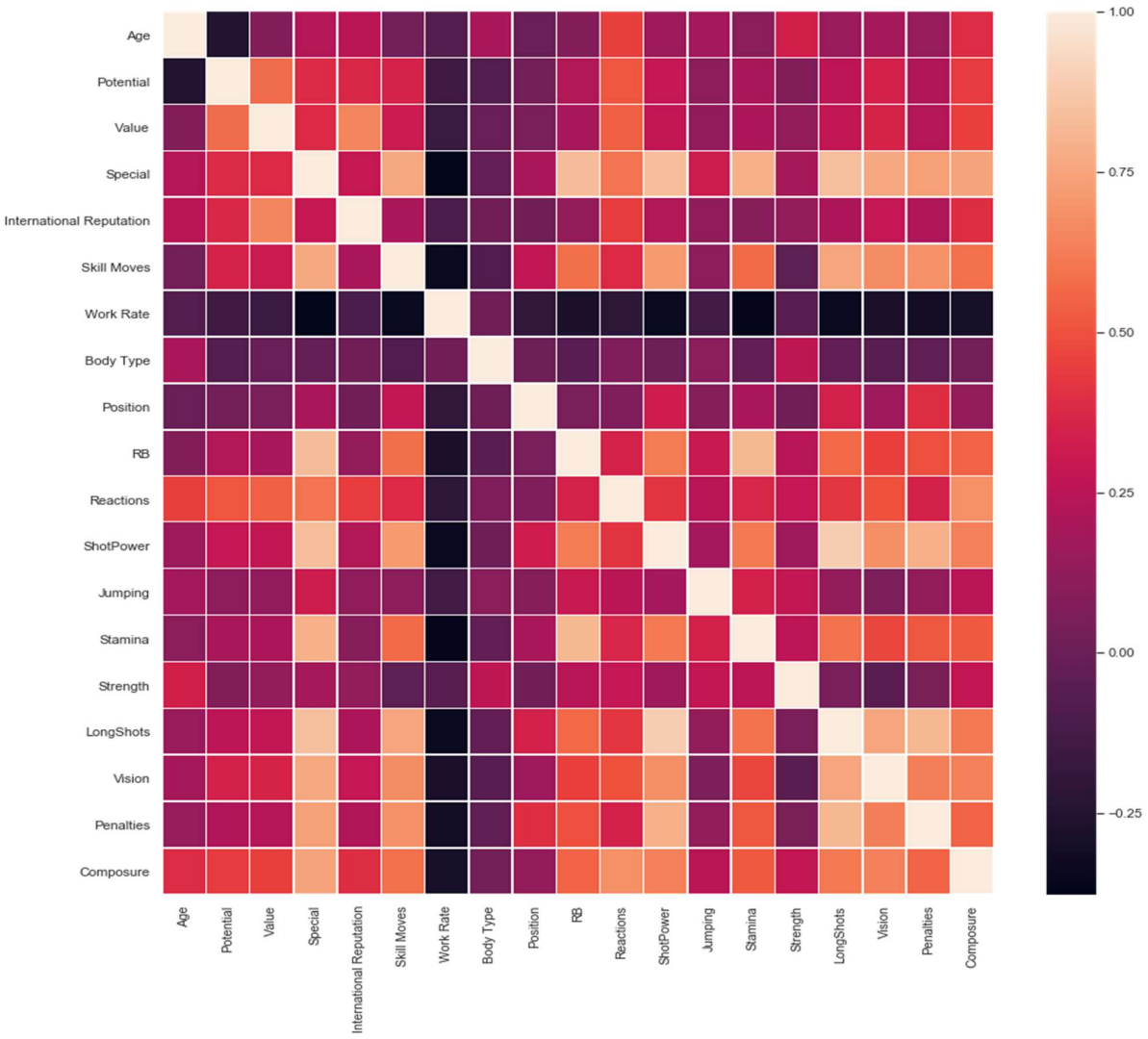
4.10.2 Columns showing high correlation coefficient are dropped to avoid Collinearity.

```
Dropping correalted features like 'ST', 'LS','RS', 'LW', 'LF', 'CF', 'RF',
'RW', 'LAM', 'CAM', 'RAM','LM', 'LCM', 'CM', 'RM', 'LWB', 'LDM', 'CDM', 'R
DM', 'RWB', 'LB','LCB', 'CB','RCB','RCM', 'GKDiving', 'GKHandling', 'GKKic
king','GKPositioning','GKReflexes'
```

4.10.2 Using heatmap checking again if collinearity is removed or note

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x194852f74e0>
```



No collinearity is left because only diagonals are light in color which has to be because each column will always be collinear to itself

Displaying Sample of processed data

Out[52]:

	Age	Potential	Value	Special	International Reputation	Skill Moves	Work Rate	Body Type	Position	RB	R
0	31	94	110500000.0	2202	5.0	4.0	8	0	21	61.0	
1	33	94	77000000.0	2228	5.0	5.0	1	1	26	64.0	
2	26	93	118500000.0	2143	5.0	5.0	2	0	14	63.0	
3	27	93	72000000.0	1471	4.0	1.0	8	0	5	11.0	
4	27	92	102000000.0	2281	4.0	4.0	0	1	19	76.0	

Columns present in the data set finally

Out[53]:

```
Index(['Age', 'Potential', 'Value', 'Special', 'International Reputation',
       'Skill Moves', 'Work Rate', 'Body Type', 'Position', 'RB', 'Reactio
ns',
       'ShotPower', 'Jumping', 'Stamina', 'Strength', 'LongShots', 'Visio
n',
       'Penalties', 'Composure'],
      dtype='object')
```

5. Columns on which the target variable depends.

Out[54]:

```
Index(['Age', 'Potential', 'Value', 'Special', 'International Reputation',
       'Skill Moves', 'Work Rate', 'Body Type', 'Position', 'RB', 'Reactio
ns',
       'ShotPower', 'Jumping', 'Stamina', 'Strength', 'LongShots', 'Visio
n',
       'Penalties', 'Composure'],
      dtype='object')
```

6. Selecting the independent variables or features for prediction.

```
'Work Rate', 'Body Type', 'Position' doesnot affect the accuracy alot so t
hey are not considered and rest all columns are considerd
Selected features: Index(['Age', 'Potential', 'Value', 'Special', 'Interna
tional Reputation',
       'Skill Moves', 'RB', 'Reactions', 'ShotPower', 'Jumping', 'Stamin
a',
       'Strength', 'LongShots', 'Vision', 'Penalties', 'Composure'],
      dtype='object')
```

7. Model fitting using LinearRegression.

7.1 Calculating the Random State.

```
The Random State for Train test split for Linear Regression: 1897
```

7.2 Splitting the data into train and test sets

```
Data and target are split into x_train,x_test,y_train,y_test for Linear
Regression
```

7.3 Fitting the Model using Linear Regression

Out[57]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

7.4 Predicting on basis of Model.

```
The Model for Linear Regression is predicted
```

7.5 Checking the Accuracy of the Model.

```
The Accuracy Score using Linear Regression is:  0.9304987566772231
```

8. Model fitting using LinearRegression.

8.1 Finding the Random State (for DecisionTreeRegressor).

```
The Random State for Train test split for DecisionTreeRegressor: 528
```

8.2 Splitting the data into train and test sets

```
Data and target are split into x_train,x_test,y_train,y_test for DecisionT
reeRegressor
```

8.3 Fitting the Model using DecisionTreeRegressor

Out[61]:

```
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=0.02,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

8.4 Predicting on basis of Model.

```
The Model for DecisionTreeRegressor is predicted
```

8.5 Checking the Accuracy of the Model.

```
The Accuracy Score using DecisionTreeRegressor is:  0.9535890113872182
```