Challenge!

The Muskets Football team is preparing for the next games season and have asked you to help clean and preprocess their data and hopefully help predict number of hits by each player.

TASK:

1. Conduct a full range cleaning of the data. Provide explanations and justifications for any actions you take.
2. Preprocess the cleaned data from task 1 above and transform it into a well behaved data.
3. Select input features for an outcome feature of HITS

Presented by:

Faith Ubara-Collins

In [1]:

```
# importing relevantslibrabries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Before I begin, lets read the dataset from my device

In [2]:

```
df = pd.read_csv('Muskets_TeamData_V2.csv')
```

```
C:\Users\MRS COLLINS\AppData\Local\Temp\ipykernel_12452\2887047926.py:1: DtypeWarning: Columns (26,29,76) ha
ve mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('Muskets_TeamData_V2.csv')
```

The error above shows the dataset contains some wrong datatype. So, I'll correct the data type as i begin cleaning the data

In [3]:
```
# A glance at the dataset

df.head()
```

Out[3]:

	ID	Name	LongName	photoUrl	playerUrl	Nationality	Age	↓OV/
0	158023	L. Messi	Lionel Messi	https://cdn.sofifa.com/players/158/023/21_60.png	http://sofifa.com/player/158023/lionel-messi/2...	Argentina	33	93.(
1	20801	Cristiano Ronaldo	C. Ronaldo dos Santos Aveiro	https://cdn.sofifa.com/players/020/801/21_60.png	http://sofifa.com/player/20801/c-ronaldo-dos-s...	Portugal	35	92.(
2	200389	J. Oblak	Jan Oblak	https://cdn.sofifa.com/players/200/389/21_60.png	http://sofifa.com/player/200389/jan-oblak/210006/	Slovenia	27	91.(
3	192985	K. De Bruyne	Kevin De Bruyne	https://cdn.sofifa.com/players/192/985/21_60.png	http://sofifa.com/player/192985/kevin-de-bruyn...	Belgium	29	91.(
4	190871	Neymar Jr	Neymar da Silva Santos Jr.	https://cdn.sofifa.com/players/190/871/21_60.png	http://sofifa.com/player/190871/neymar-da-silv...	Brazil	28	91.(

5 rows × 77 columns

Observe the shape of the dataset

In [4]:
```
df.shape
```

Out[4]: (19021, 77)

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

Out[5]:
```
Index(['ID', 'Name', 'LongName', 'photoUrl', 'playerUrl', 'Nationality', 'Age',
       '↓OVA', 'POT', 'Club', 'Contract', 'Positions', 'Height', 'Weight',
       'Preferred Foot', 'BOV', 'Best Position', 'Joined', 'Loan Date End',
       'Value', 'Wage', 'Release Clause', 'Attacking', 'Crossing', 'Finishing',
       'Heading Accuracy', 'Short Passing', 'Volleys', 'Skill', 'Dribbling',
       'Curve', 'FK Accuracy', 'Long Passing', 'Ball Control', 'Movement',
       'Acceleration', 'Sprint Speed', 'Agility', 'Reactions', 'Balance',
       'Power', 'Shot Power', 'Jumping', 'Stamina', 'Strength', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Penalties', 'Composure', 'Defending', 'Marking', 'Standing Tackle',
       'Sliding Tackle', 'Goalkeeping', 'GK Diving', 'GK Handling',
       'GK Kicking', 'GK Positioning', 'GK Reflexes', 'Total Stats',
       'Base Stats', 'W/F', 'SM', 'A/W', 'D/W', 'IR', 'PAC', 'SHO', 'PAS',
       'DRI', 'DEF', 'PHY', 'Hits'],
      dtype='object')
```

The dataset contains 19,021 records (rows) and 77 feilds (columns)

TASK 1. FULL RANGE CLEANING:

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19021 entries, 0 to 19020
Data columns (total 77 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19021 non-null  int64  
 1   Name              19021 non-null  object 
 2   LongName          19021 non-null  object 
 3   photoUrl          19021 non-null  object 
 4   playerUrl         19021 non-null  object 
 5   Nationality       19021 non-null  object 
 6   Age               19021 non-null  int64  
 7   ↓OVA              19019 non-null  float64
 8   POT               19020 non-null  float64
 9   Club              19021 non-null  object 
 10  Contract          19021 non-null  object 
 11  Positions         19021 non-null  object 
 12  Height            19021 non-null  object 
 13  Weight            19020 non-null  object 
 14  Preferred Foot    19021 non-null  object 
 15  BOV               19021 non-null  int64  
 16  Best Position     19021 non-null  object 
 17  Joined            19021 non-null  object 
 18  Loan Date End     1015 non-null   object 
 19  Value             19021 non-null  object 
 20  Wage              19021 non-null  object 
 21  Release Clause    19018 non-null  object 
 22  Attacking         19020 non-null  float64
 23  Crossing          19020 non-null  float64
 24  Finishing         19016 non-null  float64
 25  Heading Accuracy  19013 non-null  float64
 26  Short Passing     19012 non-null  object 
 27  Volleys           19014 non-null  float64
 28  Skill             19015 non-null  float64
 29  Dribbling         19020 non-null  object 
 30  Curve             19013 non-null  float64
 31  FK Accuracy       19015 non-null  float64
 32  Long Passing      19018 non-null  float64
 33  Ball Control      19018 non-null  float64
 34  Movement          19016 non-null  float64
 35  Acceleration      19017 non-null  float64
 36  Sprint Speed      19018 non-null  float64
 37  Agility           19019 non-null  float64
```

```
 38  Reactions         19017 non-null  float64
 39  Balance           19014 non-null  float64
 40  Power             19020 non-null  float64
 41  Shot Power        19019 non-null  float64
 42  Jumping           19017 non-null  float64
 43  Stamina           19020 non-null  float64
 44  Strength          19016 non-null  float64
 45  Long Shots        19014 non-null  float64
 46  Mentality         19015 non-null  float64
 47  Aggression        19021 non-null  int64
 48  Interceptions     19017 non-null  float64
 49  Positioning       19020 non-null  float64
 50  Vision            19021 non-null  int64
 51  Penalties         19020 non-null  float64
 52  Composure         19020 non-null  float64
 53  Defending         19020 non-null  float64
 54  Marking           19019 non-null  float64
 55  Standing Tackle   19018 non-null  float64
 56  Sliding Tackle    19020 non-null  float64
 57  Goalkeeping       19018 non-null  float64
 58  GK Diving         19020 non-null  float64
 59  GK Handling       19020 non-null  float64
 60  GK Kicking        19019 non-null  float64
 61  GK Positioning    19019 non-null  float64
 62  GK Reflexes       19021 non-null  int64
 63  Total Stats       19020 non-null  float64
 64  Base Stats        19021 non-null  int64
 65  W/F               19021 non-null  object
 66  SM                19021 non-null  object
 67  A/W               19021 non-null  object
 68  D/W               19020 non-null  object
 69  IR                19021 non-null  object
 70  PAC               19018 non-null  float64
 71  SHO               19018 non-null  float64
 72  PAS               19016 non-null  float64
 73  DRI               19019 non-null  float64
 74  DEF               19016 non-null  float64
 75  PHY               19020 non-null  float64
 76  Hits              16426 non-null  object
dtypes: float64(45), int64(7), object(25)
memory usage: 11.2+ MB
```

In [ ]:

```
```

Now, I will begin the sub tasks to effeciently follow my tasks.

Sub Tasks

sub task 1.

Extract the player names from the PlayerUrl column and create a new column name Player Name from the extracts

In [7]:

```
# The code below, extracts player's name from the player URL

df['Player Name'] = df['playerUrl'].str.split('/').str[-3].str.title().str.replace('-', ' ')

```

In [8]:

```
df['Player Name']
```

Out[8]:

```
0                        Lionel Messi
1           C Ronaldo Dos Santos Aveiro
2                           Jan Oblak
3                     Kevin De Bruyne
4             Neymar Da Silva Santos Jr
                     ...
19016                          Ao Xia
19017                       Ben Hough
19018                  Ronan Mckinley
19019                    Zhenao Wang
19020                      Xiao Zhou
Name: Player Name, Length: 19021, dtype: object
```

A new column " Player Name" has been created to answer sub task 1

sub task 2.

Create a new column titled Player Status from the CONTRACT column with 3 labels;

- 'Active' If the player has an active contract

- 'Free', if the player is free
- 'On Loan' if the player is on loan

In [9]:
```
df.Contract.unique()
```

Out[9]:
```
array(['2004 ~ 2021', '2018 ~ 2022', '2014 ~ 2023', '2015 ~ 2023',
       '2017 ~ 2022', '2017 ~ 2023', '2018 ~ 2024', '2014 ~ 2022',
       '2018 ~ 2023', '2016 ~ 2023', '2013 ~ 2023', '2011 ~ 2023',
       '2009 ~ 2022', '2005 ~ 2021', '2011 ~ 2021', '2015 ~ 2022',
       '2017 ~ 2024', '2010 ~ 2024', '2012 ~ 2021', '2019 ~ 2024',
       '2015 ~ 2024', '2017 ~ 2025', '2020 ~ 2025', '2019 ~ 2023',
       '2008 ~ 2023', '2015 ~ 2021', '2020 ~ 2022', '2012 ~ 2022',
       '2016 ~ 2025', '2013 ~ 2022', '2011 ~ 2022', '2012 ~ 2024',
       '2016 ~ 2021', '2012 ~ 2023', '2008 ~ 2022', '2019 ~ 2022',
       '2017 ~ 2021', '2013 ~ 2024', '2020 ~ 2024', '2010 ~ 2022',
       '2020 ~ 2021', '2011 ~ 2024', '2020 ~ 2023', '2014 ~ 2024',
       '2013 ~ 2026', '2016 ~ 2022', '2010 ~ 2021', '2013 ~ 2021',
       '2019 ~ 2025', '2018 ~ 2025', '2016 ~ 2024', '2018 ~ 2021',
       '2009 ~ 2024', '2007 ~ 2022', 'Jun 30, 2021 On Loan',
       '2009 ~ 2021', '2019 ~ 2021', '2019 ~ 2026', 'Free', '2012 ~ 2028',
       '2010 ~ 2023', '2014 ~ 2021', '2015 ~ 2025', '2014 ~ 2026',
       '2012 ~ 2025', '2017 ~ 2020', '2002 ~ 2022', '2020 ~ 2027',
       '2013 ~ 2025', 'Dec 31, 2020 On Loan', '2019 ~ 2020',
       '2011 ~ 2025', '2016 ~ 2020', '2007 ~ 2021', '2020 ~ 2026',
       '2010 ~ 2025', '2009 ~ 2023', '2008 ~ 2021', '2020 ~ 2020',
       '2016 ~ 2026', 'Jan 30, 2021 On Loan', '2012 ~ 2020',
       '2014 ~ 2025', 'Jun 30, 2022 On Loan', '2015 ~ 2020',
       'May 31, 2021 On Loan', '2018 ~ 2020', '2014 ~ 2020',
       '2013 ~ 2020', '2006 ~ 2024', 'Jul 5, 2021 On Loan',
       'Dec 31, 2021 On Loan', '2004 ~ 2025', '2011 ~ 2020',
       'Jul 1, 2021 On Loan', 'Jan 1, 2021 On Loan', '2006 ~ 2023',
       'Aug 31, 2021 On Loan', '2006 ~ 2021', '2005 ~ 2023',
       '2003 ~ 2020', '2009 ~ 2020', '2002 ~ 2020', '2005 ~ 2020',
       '2005 ~ 2022', 'Jan 31, 2021 On Loan', '2010 ~ 2020',
       'Dec 30, 2021 On Loan', '2008 ~ 2020', '2007 ~ 2020',
       '2003 ~ 2021', 'Jun 23, 2021 On Loan', 'Jan 3, 2021 On Loan',
       'Nov 27, 2021 On Loan', '2002 ~ 2021', 'Jan 17, 2021 On Loan',
       'Jun 30, 2023 On Loan', '1998 ~ 2021', '2003 ~ 2022',
       '2007 ~ 2023', 'Jul 31, 2021 On Loan', 'Nov 22, 2020 On Loan',
       'May 31, 2022 On Loan', '2006 ~ 2020', 'Dec 30, 2020 On Loan',
       '2007 ~ 2025', 'Jan 4, 2021 On Loan', 'Nov 30, 2020 On Loan',
       '2004 ~ 2020', '2009 ~ 2025', 'Aug 1, 2021 On Loan'], dtype=object)
```

In [10]:

```
# This code will help categorize the Contract column into 3 categories and add those new categories at Pl

status = []
for a in df['Contract']:
    if 'On Loan' in a:
        status.append('On Loan')
    elif 'Free' in a:
        status.append('Free')
    else:
        status.append('Active')
df['Player Status'] = status
```

Checking the see each new player's status

In [11]:

```
# When it is on loan

df[df['Player Status'] == 'On Loan'][['Contract', 'Player Status']]
```

Out[11]:

	Contract	Player Status
205	Jun 30, 2021 On Loan	On Loan
248	Jun 30, 2021 On Loan	On Loan
254	Jun 30, 2021 On Loan	On Loan
302	Jun 30, 2021 On Loan	On Loan
306	Jun 30, 2021 On Loan	On Loan
...	...	...
18514	Aug 31, 2021 On Loan	On Loan
18613	Jun 30, 2021 On Loan	On Loan
18642	Dec 31, 2020 On Loan	On Loan
18664	Dec 31, 2020 On Loan	On Loan
18722	Dec 31, 2020 On Loan	On Loan

1015 rows × 2 columns

We have Exactly 1,015 players currently on Loan

In [12]:
```
# To see the players that are active

df[df['Player Status'] == 'Active'][['Contract', 'Player Status']]
```

Out[12]:

	Contract	Player Status
0	2004 ~ 2021	Active
1	2018 ~ 2022	Active
2	2014 ~ 2023	Active
3	2015 ~ 2023	Active
4	2017 ~ 2022	Active
...	...	...
19016	2018 ~ 2022	Active
19017	2020 ~ 2021	Active
19018	2019 ~ 2020	Active
19019	2020 ~ 2022	Active
19020	2019 ~ 2023	Active

17769 rows × 2 columns

The active players are 17,769; which is the highest number of players

In [13]:
```python
# Check the players that are free

df[df['Player Status'] == 'Free'][['Contract', 'Player Status']]
```

Out[13]:

	Contract	Player Status
289	Free	Free
292	Free	Free
369	Free	Free
374	Free	Free
375	Free	Free
...	...	...
17262	Free	Free
17385	Free	Free
17701	Free	Free
17703	Free	Free
18247	Free	Free

237 rows × 2 columns

The display shows that only 237 players are free

In [ ]:
```python

```

sub task 3.

Unpack the POSITIONS column into as many columns as there are possible.

- Assign Boolean values in the columns for each player as appropriate.
- Name the columns the play position.

In [14]:
```
df['Positions'].unique()
```

Out[14]:
```
array(['RW, ST, CF', 'ST, LW', 'GK', 'CAM, CM', 'LW, CAM', 'ST', 'RW',
       'ST, LW, RW', 'CB', 'LW', 'CDM', 'CF, ST', 'LW, RW', 'CDM, CM',
       'CDM, RB', 'CF, CAM', 'LW, ST', 'CM', 'ST, CF, LW', 'RM, LM, CAM',
       'RB', 'RW, CAM, CM', 'LB', 'LM, CF', 'CF', 'RW, LW', 'CAM, RM, RW',
       'CM, CDM', 'CAM, CF, ST', 'CM, CDM, CAM', 'CF, LW, CAM',
       'CAM, RM, CF', 'LM, ST', 'RM, LM, RW', 'LM', 'CAM, RW', 'CB, CDM',
       'RW, RM', 'LW, CF', 'CM, RM, LM', 'LB, LM', 'CAM, CM, RM',
       'CAM, CM, CF', 'CAM, CF', 'LM, RM, LW', 'LM, LB, CM', 'CM, LM, LB',
       'RM, RW', 'RM, CM', 'CAM, CM, LW', 'CB, LB', 'RM, RB', 'ST, RW',
       'LM, RW, LW', 'RB, LB', 'RB, RM', 'RM', 'LM, RM, CF', 'CAM, RM',
       'RB, RWB', 'CDM, CB, CM', 'CAM, RM, ST', 'LM, LW, RM', 'CM, CAM',
       'ST, RM, CF', 'LM, RM', 'RM, CF', 'LM, LWB', 'RW, RM, CF',
       'RB, CM', 'LW, CAM, RW', 'CAM, LW, CM', 'CM, CAM, CDM',
       'RW, LW, CAM', 'CM, CAM, LM', 'CM, RM, ST', 'CDM, CM, RB',
       'ST, CAM', 'CAM, LW, ST', 'LB, CB, LWB', 'RM, ST', 'CB, CDM, LB',
       'RWB, RM', 'CM, LM, RM', 'RB, CDM, CM', 'RW, LW, RM', 'LM, LW',
       'CM, LM', 'LM, LB', 'RM, LM, CF', 'LB, LM, RM', 'CDM, CM, CAM',
       'ST, LW, RM', 'CAM, CM, ST', 'ST, CF', 'LWB, LB', 'LW, RW, LM',
       'RM, RW, ST', 'LWB', 'CF, ST, CAM', 'LM, CAM, RM', 'RB, CB',
```

In [15]:
```
df.Positions.values
```

Out[15]:
```
array(['RW, ST, CF', 'ST, LW', 'GK', ..., 'CM', 'RW', 'CB, LB'],
      dtype=object)
```

The code below will help me solve sub task 3;

- This code will split the positions in the 'POSITIONS' column into separate columns,
- assign (1 and 0) Boolean values indicating whether each player has that position.
- The columns will be named 'Position_' followed by the play position.***

In [16]:
```python

# Split the positions into separate columns with Boolean values
positions = df['Positions'].str.get_dummies(sep=', ')

# Assign column names based on play position(where 1 = True and 0 = False)
positions.columns = ['position_' + col for col in positions.columns]

# Concatenate the original DataFrame with the positions columns
df = pd.concat([df, positions], axis=1)


```

In [17]:
```python
# Check the newly created columns

positions.columns
```

Out[17]:
```
Index(['position_CAM', 'position_CB', 'position_CDM', 'position_CF',
       'position_CM', 'position_GK', 'position_LB', 'position_LM',
       'position_LW', 'position_LWB', 'position_RB', 'position_RM',
       'position_RW', 'position_RWB', 'position_ST'],
      dtype='object')
```

Those are the newly created columns from the POSITIONS column

Demonstrate how the outcome will look:

- where the play position is found, it returns 1 for True
- Where Not found, it returns 0 for False

In [18]:
```
df[['Positions'] + list(positions.columns)].head()
```

Out[18]:

	Positions	position_CAM	position_CB	position_CDM	position_CF	position_CM	position_GK	position_LB	position_LM	position_LW
0	RW, ST, CF	0	0	0	1	0	0	0	0	0
1	ST, LW	0	0	0	0	0	0	0	0	1
2	GK	0	0	0	0	0	1	0	0	0
3	CAM, CM	1	0	0	0	1	0	0	0	0
4	LW, CAM	1	0	0	0	0	0	0	0	1

In [ ]:
```

```

sub task 4.

Weight and Height, W/F, SM and IR Columns: convert to integers.

But before I change the datatype, i need to confirm that all values in each column can be intergers

- *Reasons because, data tyoe int cannot contain string, 'kg, lbs and stars*

In [19]:
```
df.SM.unique()
```

Out[19]:
```
array(['4★', '5★', '1★', '2★', '3★'], dtype=object)
```

In [20]:
```
df.IR.unique()
```

Out[20]:
```
array(['5 ★', '3 ★', '4 ★', '2 ★', '1 ★'], dtype=object)
```

In [21]:
```
df['W/F'].unique()
```

Out[21]:
```
array(['4 ★', '3 ★', '5 ★', '2 ★', '1 ★'], dtype=object)
```

From the above unique values, it is important that we remove the star before converting the column to int

In [22]:
```
# By using the caret (^), it ensures that only the digits at the beginning of the string are extracted.

df['IR']= df['IR'].str.extract('^(\d+)')
df['SM'] = df['IR'].str.extract('^(\d+)')
df['W/F'] = df['IR'].str.extract('^(\d+)')
```

In [23]:
```
# Check the new unique values


columns_to_check = ['W/F', 'SM', 'IR']  # List of columns to check

unique_values = df[columns_to_check].apply(lambda x: x.unique())
print(unique_values)

```

```
  W/F SM IR
0   5  5  5
1   3  3  3
2   4  4  4
3   2  2  2
4   1  1  1
```

I will also check for the weight and heigth column

In [24]:
```
df['Weight'].unique()
```

Out[24]:
```
array(['72kg', '83kg', '87kg', '70kg', '68kg', '80kg', '71kg', '91kg',
       '73kg', '85kg', '92kg', '69kg', '84kg', '96kg', '81kg', '82kg',
       '75kg', '86kg', '89kg', '74kg', '76kg', '64kg', '78kg', '90kg',
       '66kg', '60kg', '94kg', '79kg', '67kg', '65kg', '59kg', '61kg',
       '93kg', '88kg', '97kg', '77kg', '62kg', '63kg', '95kg', '100kg',
       nan, '58kg', '183lbs', '179lbs', '172lbs', '196lbs', '176lbs',
       '185lbs', '170lbs', '203lbs', '168lbs', '161lbs', '146lbs',
       '130lbs', '190lbs', '174lbs', '148lbs', '165lbs', '159lbs',
       '192lbs', '181lbs', '139lbs', '154lbs', '157lbs', '163lbs', '98kg',
       '103kg', '99kg', '102kg', '56kg', '101kg', '57kg', '55kg', '104kg',
       '107kg', '110kg', '53kg', '50kg', '54kg', '52kg'], dtype=object)
```

I will convert all "lbs" to kg for consistency

In [25]:
```
# This code will convert all lbs to kg and also remove kg from all items so that we can chnage the data t

df['Weight'] = df['Weight'].apply(lambda x: float(x.replace("lbs", "")) * 0.45359237 if isinstance(x, str
                                       and "lbs" in x else x)

df['Weight']= df['Weight'].replace('kg', ' ', regex=True)

df.Weight.unique()
```

Out[25]:
```
array(['72 ', '83 ', '87 ', '70 ', '68 ', '80 ', '71 ', '91 ', '73 ',
       '85 ', '92 ', '69 ', '84 ', '96 ', '81 ', '82 ', '75 ', '86 ',
       '89 ', '74 ', '76 ', '64 ', '78 ', '90 ', '66 ', '60 ', '94 ',
       '79 ', '67 ', '65 ', '59 ', '61 ', '93 ', '88 ', '97 ', '77 ',
       '62 ', '63 ', '95 ', '100 ', nan, '58 ', 83.00740371,
       81.19303423000001, 78.01788764, 88.90410452, 79.83225712000001,
       83.91458845000001, 77.1107029, 92.07925111, 76.20351816,
       73.02837157, 66.22448602, 58.9670081, 86.1825503, 78.92507238,
       67.13167076, 74.84274105, 72.12118683, 87.08973504000001,
       82.10021897, 63.04933943, 69.85322498000001, 71.21400209000001,
       73.93555631000001, '98 ', '103 ', '99 ', '102 ', '56 ', '101 ',
       '57 ', '55 ', '104 ', '107 ', '110 ', '53 ', '50 ', '54 ', '52 '],
      dtype=object)
```

Now that I've fixed the Weight, I'll move on to the Height

In [26]:
```
df['Height'].unique()
```

Out[26]:
```
array(['170cm', '187cm', '188cm', '181cm', '175cm', '184cm', '191cm',
       '178cm', '193cm', '185cm', '199cm', '173cm', '168cm', '176cm',
       '177cm', '183cm', '180cm', '189cm', '179cm', '195cm', '172cm',
       '182cm', '186cm', '192cm', '165cm', '194cm', '167cm', '196cm',
       '163cm', '190cm', '174cm', '169cm', '171cm', '197cm', '200cm',
       '166cm', '6\'2"', '164cm', '198cm', '6\'3"', '6\'5"', '5\'11"',
       '6\'4"', '6\'1"', '6\'0"', '5\'10"', '5\'9"', '5\'6"', '5\'7"',
       '5\'4"', '201cm', '158cm', '162cm', '161cm', '160cm', '203cm',
       '157cm', '156cm', '202cm', '159cm', '206cm', '155cm'], dtype=object)
```

The method employed here, will check if the string contains a single quote ('). If it does,

- *it splits the value into feet and inches*

- *converts them to centimeters using the conversion factors of 30.48 and 2.54 respectively.*

In [27]:
```
# Now all the values will be in cm, even though i remove the cm.

#df['Height'] = df['Height'].apply(lambda x: int(x.split("'")[0]) * 30.48 +
                                       #int(x.split("'")[1].replace('"', '')) * 2.54 if "'" in x else x)

df['Height'] = df['Height'].apply(lambda x:
                                  int(x.split("'")[0]) * 30.48 + int(x.split("'")[1].replace('"', '')) *
                                  2.54 if isinstance(x, str) and "'" in x else x)

df['Height']= df['Height'].replace('cm', '', regex=True)

df.Height.unique()
```

Out[27]:
```
array(['170', '187', '188', '181', '175', '184', '191', '178', '193',
       '185', '199', '173', '168', '176', '177', '183', '180', '189',
       '179', '195', '172', '182', '186', '192', '165', '194', '167',
       '196', '163', '190', '174', '169', '171', '197', '200', '166',
       187.96, '164', '198', 190.5, 195.57999999999998, 180.34, 193.04,
       185.42, 182.88, 177.8, 175.26, 167.64000000000001, 170.18, 162.56,
       '201', '158', '162', '161', '160', '203', '157', '156', '202',
       '159', '206', '155'], dtype=object)
```

In [ ]:
```

```

Now we I can complete the sub task 4

- converting the columns below to integers

In [28]:
```
df['Weight'].isnull().sum()
```

Out[28]:
```
1
```

In [29]:
```python
# I will be converting the data types into intergers
df['Weight'].fillna(0, inplace=True) # the Wieght contains 1 missing value

df['Weight'] = df['Weight'].astype('int64')

df['Height'] = df['Height'].astype('int64')
df['W/F'] = df['W/F'].astype('int64')
df['SM'] = df['SM'].astype('int64')
df['IR'] = df['IR'].astype('int64')
```

In [30]:
```python
# Let's confrim that the data type has changed

columns_to_check = ['Weight', 'Height', 'W/F', 'SM', 'IR']  # List of columns to check

data_types = df[columns_to_check].dtypes
print(data_types)
```

```
Weight    int64
Height    int64
W/F       int64
SM        int64
IR        int64
dtype: object
```

In [ ]:
```python

```

sub task 5.

Value, Wage and Release Clause columns: convert to Float

In [31]:

```
df[['Wage', 'Value', 'Release Clause']]
```

Out[31]:

	Wage	Value	Release Clause
0	€560K	€103.5M	€138.4M
1	€220K	€63M	€75.9M
2	€125K	€120M	€159.4M
3	€370K	€129M	€161M
4	€270K	€132M	€166.5M
...	...	...	...
19016	€1K	€100K	€70K
19017	€500	€130K	€165K
19018	€500	€120K	€131K
19019	€2K	€100K	€88K
19020	€1K	€100K	€79K

19021 rows × 3 columns

The first and second function checks if the string contains 'K' and 'M'. If it does, it removes the 'K' and 'M' character using replace() and converts the remaining value to a float. Then, it multiplies the value by 1000 and 1000000 respectively to convert it to a thousand and million as the case may be...

In [32]:

```
df['Value'] = df['Value'].apply(lambda x: float(x.replace('K', '').replace('€', '')) * 1000 if 'K' in str
df['Value'] = df['Value'].apply(lambda x: float(x.replace('M', '').replace('€', '')) * 1000000 if 'M' in
df['Value']= df['Value'].replace('€', '', regex=True)

```

Same thing goes for Wage and Release Clause columns

In [33]:
```
# converting each value to thousand or million and leaving all in In Euro
df['Wage'] = df['Wage'].apply(lambda x: float(x.replace('K', '').replace('€', '')) * 1000 if 'K' in str(x
df['Wage'] = df['Wage'].apply(lambda x: float(x.replace('M', '').replace('€', '')) * 1000000 if 'M' in st
df['Wage']= df['Wage'].replace('€', '', regex=True)

```

In [34]:
```
df['Release Clause']= df['Release Clause'].apply(lambda x:
                                                float(x.replace('K', '').replace('€', '')) * 1000 if 'K'
df['Release Clause']=df['Release Clause'].apply(lambda x:
                                            float(x.replace('M', '').replace('€', '')) * 1000000 if 'M' i
df['Release Clause']= df['Release Clause'].replace('€', '', regex=True)

```

Check if the changes has been implimented

In [35]:
```
# Return the table for only thses columns

df[['Wage', 'Value', 'Release Clause']]
```

Out[35]:

	Wage	Value	Release Clause
0	560000.0	103500000.0	138400000.0
1	220000.0	63000000.0	75900000.0
2	125000.0	120000000.0	159400000.0
3	370000.0	129000000.0	161000000.0
4	270000.0	132000000.0	166500000.0
...	...	...	...
19016	1000.0	100000.0	70000.0
19017	500	130000.0	165000.0
19018	500	120000.0	131000.0
19019	2000.0	100000.0	88000.0
19020	1000.0	100000.0	79000.0

19021 rows × 3 columns

In [36]:
```
# converting them to float

df['Wage'] = df['Wage'].astype('float')
df['Value'] = df['Value'].astype('float')
df['Release Clause'] = df['Release Clause'].astype('float')

```

Check the updated datatypes

In [37]:
```
columns_to_check = ['Wage', 'Value', 'Release Clause']  # List of columns to check

data_types = df[columns_to_check].dtypes
print(data_types)
```

```
Wage              float64
Value             float64
Release Clause    float64
dtype: object
```

Sub task 5 successfully completed

sub task 6.

Inspect the HITS column and ensure its float

In [38]:
```
# To begin,  I looked at the unique values

df['Hits'].unique()
```

Out[38]:
```
array(['771', '562', '150', '207', '595', '248', '246', '120', '1.6K',
       '130', '321', '189', '175', '96', '118', '216', '212', '154',
       '205', '202', '339', '408', '103', '332', '86', '173', '161',
       '396', '1.1K', '433', '242', '206', '177', '1.5K', '198', '459',
       '117', '119', '209', '84', '187', '165', '203', '65', '336', '126',
       '313', '124', '145', '538', '182', '101', '45', '377', '99', '194',
       '403', '414', '593', '374', '245', '3.2K', '266', '299', '309',
       '215', '265', '211', '112', '337', '70', '159', '688', '116', '63',
       '144', '123', '71', '224', '113', '168', '61', '89', '137', '278',
       '75', '148', '176', '197', '264', '214', '247', '402', '440',
       '1.7K', '2.3K', '171', '320', '657', '87', '259', '200', '255',
       '253', '196', '60', '97', '85', '169', '256', '132', '239', '166',
       '121', '109', '32', '46', '122', '48', '527', '199', '282', '51',
       '1.9K', '642', '155', '323', '288', '497', '509', '79', '49',
       '270', '511', '80', '128', '115', '156', '204', '143', '140',
       '152', '220', '134', '225', '94', '74', '135', '142', '50', '77',
       '40', '107', '193', '179', '34', '64', '453', '57', '81', '28',
       '78', '133', '43', '425', '88', '42', '36', '233', '376', '210',
       '444', '100', '263', '98', '29', '160', '39', '257', '6', '310',
       '138', '62', '293', '285', '362', '66', '69', '58', '21', '20',
       '131', '38', '406', '68', '108', '110', '93', '512', '443', '306',
       '352', '422', '585', '346', '178', '841', '76', '394', '72', '172',
       '44', '407', '230', '367', '295', '157', '243', '56', '111', '326',
       '679', '18', '92', '59', '25', '184', '53', '12', '90', '55', '73',
       '11', '566', '180', '83', '262', '17', '26', '31', '280', '359',
       '213', '297', '387', '480', '381', '677', '486', '8', '244', '129',
       '388', '275', '319', '2K', '52', '91', '421', '153', '27', '41',
       '222', '35', '102', '23', '30', '33', '146', '13', '19', '14',
       '106', '276', '568', '353', '47', '478', '249', '254', '369',
       '219', '565', '237', '227', '434', '375', '162', '605', '654', '3',
       '7', '9', '104', '114', '186', '446', '756', '22', '139', '500',
       '67', '147', '149', '16', '82', '54', '37', '15', '1.3K', '3K',
       '952', '5', '749', '541', '330', '393', '517', '770', '409', '170',
       '125', '283', '342', '363', '580', '105', '217', '24', '141', '10',
       '427', '158', '426', '4', '666', '181', '324', '979', '1.4K',
       '302', '751', '298', '411', '944', '2', '947', '292', '349', '621',
       '1', '2.8K', '338', '287', '261', '218', '1.8K', '240', '279',
       '229', '188', '315', '664', '613', '190', '706', '127', '462',
       '386', '695', '491', '167', '281', '250', '307', '95', '231',
       '174', '680', '633', '221', '348', '602', '183', '653', '195',
       '164', '151', '258', '8.4K', '343', '419', '655', '136', '399',
       '531', '357', '228', '385', '312', '340', '238', '487', '355',
       '499', '4.3K', '296', '515', '943', '1.2K', '903', '335', '191',
```

```
       '594', '267', '617', '516', '504', '331', '652', '410', '550',
       '473', '442', '344', '208', '1K', '2.5K', '273', '485', '826',
       '192', '405', '941', '477', '644', '303', '417', '6K', nan, 1.0,
       5.0, 9.0, 2.0, 21.0, 7.0, 3.0, 12.0, 6.0, 4.0, 8.0, 13.0, 11.0,
       31.0, 10.0, 17.0], dtype=object)
```

From the display above, I noticed that some values had "K" which indicate 1,000 and the date type is object(str)

- First we remove the K: mutiply the values with k by 1,000.
- Then change the data type to float

In [39]:

```
# this code removes K
#df['Hits']= df['Hits'].replace('K', '', regex=True)
df['Hits'] = df['Hits'].apply(lambda x: float(x[:-1])*1000 if 'K' in str(x) else x)

# this code changes the data type
df['Hits'] = df['Hits'].astype('float')
```

In [40]:

```
df['Hits'].dtype
```

Out[40]:

```
dtype('float64')
```

In [41]:

```
df['Hits']
```

Out[41]:

```
0        771.0
1        562.0
2        150.0
3        207.0
4        595.0
         ...
19016      NaN
19017      NaN
19018      NaN
19019      NaN
19020      NaN
Name: Hits, Length: 19021, dtype: float64
```

In [42]:
```
df['Hits'].isna().sum()
```

Out[42]:
```
2595
```

In [ ]:
```

```

sub task 7.

Create 5 new categorical columns for the Height, Weight, Release Clause, Value and Wage into which you convert the respective values into clusters/labels as follows

a. Height: Bucket intervals of 10 cm

b. Weight: Bucket intervals of 10 kg

c. Wage: bucket intervals of 50K

d. Value: bucket intervals of 50M

e. Release Clause: bucket intervals of 50M

To begin:

Check each column for their maximum and minimum numbers

In [43]:

```python
# Specify the columns you want to calculate the max and min values for
columns_to_analyze = ['Height', 'Weight', 'Wage', 'Value', 'Release Clause']

# Calculate the maximum and minimum values for the specified columns
max_values = df[columns_to_analyze].max()
min_values = df[columns_to_analyze].min()

# Print the maximum and minimum values for each column
for column in columns_to_analyze:
    print(f'Max value for {column}: {max_values[column]}')
    print(f'Min value for {column}: {min_values[column]}')
    print('--------------------')

```

```
Max value for Height: 206.0
Min value for Height: 155.0
--------------------
Max value for Weight: 110.0
Min value for Weight: 0.0
--------------------
Max value for Wage: 560000.0
Min value for Wage: 0.0
--------------------
Max value for Value: 185500000.0
Min value for Value: 0.0
--------------------
Max value for Release Clause: 203100000.0
Min value for Release Clause: 0.0
--------------------
```

In [44]:
```
# Define the binning intervals and custom labels for each variable

height_intervals = [0, 165, 175, 185,195,205,215]
height_labels = ['Very Short','Short', 'Below Average','Average', 'Tall', 'Very Tall']

weight_intervals = [0, 60, 70, 80, 90,100, 110]
weight_labels = ['Very Light','Light', 'Below Average','Average' ,'Heavy', 'Very Heavy']

wage_intervals = [0, 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000, 6000
wage_labels = ['Very Low','Low','Below Moderate', 'Moderate', 'Average','Above Average','Extremely', 'Hig
               'Very High','Exceptionally','Outstandingly']

value_intervals = [0, 50000000, 100000000, 150000000, 200000000]
value_labels = ['Small', 'Moderate', 'Big', 'Large']

release_intervals = [0, 50000000, 100000000, 150000000, 200000000, 250000000]
release_labels = ['Beginner', 'Novice', 'Intermediate', 'Advance', 'Expert']

# Create the categorical columns using cut() with specified labels for each variable
df['Height Category'] = pd.cut(df['Height'], bins=height_intervals, labels=height_labels, right=False)
df['Weight Category'] = pd.cut(df['Weight'], bins=weight_intervals, labels=weight_labels, right=False)
df['Wage Category'] = pd.cut(df['Wage'], bins=wage_intervals, labels=wage_labels, right=False)
df['Value Category'] = pd.cut(df['Value'], bins=value_intervals, labels=value_labels, right=False)
df['Release Category'] = pd.cut(df['Release Clause'], bins=release_intervals, labels=release_labels, righ


```

Display the updated DataFrame

In [45]:
```
df[['Height Category','Weight Category','Wage Category','Value Category', 'Release Category']]
```

Out[45]:

	Height Category	Weight Category	Wage Category	Value Category	Release Category
0	Short	Below Average	Outstandingly	Big	Intermediate
1	Average	Average	Moderate	Moderate	Novice
2	Average	Average	Low	Big	Advance
3	Below Average	Below Average	Extremely	Big	Advance
4	Below Average	Light	Average	Big	Advance
...	...	...	...	...	...
19016	Below Average	Light	Very Low	Small	Beginner
19017	Below Average	Light	Very Low	Small	Beginner
19018	Below Average	Below Average	Very Low	Small	Beginner
19019	Below Average	Light	Very Low	Small	Beginner
19020	Average	Below Average	Very Low	Small	Beginner

19021 rows × 5 columns

In [46]:
```
columns_to_check = ['Height Category', 'Weight Category', 'Wage Category',  'Release Category']  # List o

data_types = df[columns_to_check].dtypes
print(data_types)
```

```
Height Category     category
Weight Category     category
Wage Category       category
Release Category    category
dtype: object
```

Sub task 7 successfully done

In [ ]:

```python


```

Next I will check for DUPLICATES

In [47]:

```python
#  Handle duplicates, the code sums all duplicates

print("Duplicates in dataset:", df.duplicated().sum())
```

```
Duplicates in dataset: 42
```

The dataset contains 42 duplicates, so I'll drop them off.

In [48]:

```python
# In droping the duplicates, I'll keep the first one that appeared in the dataset

df = df.drop_duplicates(keep = 'first')

print("Duplicates in dataset:", df.duplicated().sum())
```

```
Duplicates in dataset: 0
```

No duplicates in our dataset again

Progress :

- I'll drop off come irrelevants columns before moving forward.
- Note this columns are dropped because i will not be needing any information in them;
- Columns that have bean unpacked and categorized
- Also to allow my model select the best features

In [49]:
```
df.columns
```

Out[49]:
```
Index(['ID', 'Name', 'LongName', 'photoUrl', 'playerUrl', 'Nationality', 'Age',
       '↓OVA', 'POT', 'Club', 'Contract', 'Positions', 'Height', 'Weight',
       'Preferred Foot', 'BOV', 'Best Position', 'Joined', 'Loan Date End',
       'Value', 'Wage', 'Release Clause', 'Attacking', 'Crossing', 'Finishing',
       'Heading Accuracy', 'Short Passing', 'Volleys', 'Skill', 'Dribbling',
       'Curve', 'FK Accuracy', 'Long Passing', 'Ball Control', 'Movement',
       'Acceleration', 'Sprint Speed', 'Agility', 'Reactions', 'Balance',
       'Power', 'Shot Power', 'Jumping', 'Stamina', 'Strength', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Penalties', 'Composure', 'Defending', 'Marking', 'Standing Tackle',
       'Sliding Tackle', 'Goalkeeping', 'GK Diving', 'GK Handling',
       'GK Kicking', 'GK Positioning', 'GK Reflexes', 'Total Stats',
       'Base Stats', 'W/F', 'SM', 'A/W', 'D/W', 'IR', 'PAC', 'SHO', 'PAS',
       'DRI', 'DEF', 'PHY', 'Hits', 'Player Name', 'Player Status',
       'position_CAM', 'position_CB', 'position_CDM', 'position_CF',
       'position_CM', 'position_GK', 'position_LB', 'position_LM',
       'position_LW', 'position_LWB', 'position_RB', 'position_RM',
       'position_RW', 'position_RWB', 'position_ST', 'Height Category',
       'Weight Category', 'Wage Category', 'Value Category',
       'Release Category'],
      dtype='object')
```

In [50]:
```
df['GK Reflexes'].unique()
```

Out[50]:
```
array([ 8, 11, 90, 13, 10, 14, 89,  6, 12, 88,  7,  9, 15,  5,  3, 37, 85,
       86,  4, 16, 82, 83, 84, 87, 78, 80, 20, 18, 79, 81, 19, 77, 17,  2,
       74, 71, 76, 73, 75, 72, 69, 46, 66, 51, 70, 34, 67, 23, 68, 45, 65,
       21, 59, 54, 47, 61, 64, 63, 62, 60, 58, 56, 57, 55, 53, 50, 52, 49,
       48, 44], dtype=int64)
```

At this point, I will be dropping off some redundant columns

In [51]:

```

col_to_drop = ['ID', 'Name', 'LongName', 'photoUrl', 'playerUrl', 'Nationality',
               '↓OVA', 'POT', 'Club','BOV', 'Best Position','Contract', 'Positions',
               'Joined', 'Loan Date End','Value', 'Wage', 'Release Clause','Height',
               'Weight','W/F', 'SM', 'A/W', 'D/W', 'IR', 'PAC', 'SHO', 'PAS','DRI', 'DEF', 'PHY','Player


# Drop the columns listed
df_m = df.loc[:, ~df.columns.isin(col_to_drop)]

# view the new dataset
df_m.head()
```

Out[51]:

	Age	Preferred Foot	Attacking	Crossing	Finishing	Heading Accuracy	Short Passing	Volleys	Skill	Dribbling	...	position_RB	position_RM	position_
0	33	Left	429.0	85.0	95.0	70.0	91	88.0	470.0	96	...	0	0	
1	35	Right	437.0	84.0	95.0	90.0	82	86.0	414.0	88	...	0	0	
2	27	Right	95.0	13.0	11.0	15.0	43	13.0	109.0	12	...	0	0	
3	29	Right	407.0	94.0	82.0	55.0	94	82.0	441.0	88	...	0	0	
4	28	Right	408.0	85.0	87.0	62.0	87	87.0	448.0	95	...	0	0	

5 rows × 67 columns

My new dataset now contains 67 columns, So i will be building my model with these 67 features

Treating the missing values

I will Use the forward and backward fill method appraoch so that:

In [52]:

```
null_counts = df_m.isnull().sum()
columns_with_null = null_counts[null_counts >= 0]
for column_name, count in columns_with_null.items():
    #print(f"Column '{column_name}' has {count} null value(s).")
    print(f' {column_name} : {count}')
```

```
Age : 0
Preferred Foot : 0
Attacking : 1
Crossing : 1
Finishing : 5
Heading Accuracy : 8
Short Passing : 9
Volleys : 7
Skill : 6
Dribbling : 1
Curve : 8
FK Accuracy : 6
Long Passing : 3
Ball Control : 3
Movement : 5
Acceleration : 4
Sprint Speed : 3
Agility : 2
Reactions : 4
Balance : 7
Power : 1
Shot Power : 2
Jumping : 4
Stamina : 1
Strength : 5
Long Shots : 7
Mentality : 6
Aggression : 0
Interceptions : 4
Positioning : 1
Vision : 0
Penalties : 1
Composure : 1
Defending : 1
Marking : 2
Standing Tackle : 3
Sliding Tackle : 1
Goalkeeping : 3
GK Diving : 1
GK Handling : 1
GK Kicking : 2
GK Positioning : 2
GK Reflexes : 0
```

```
Total Stats : 1
Base Stats : 0
Hits : 2595
Player Status : 0
position_CAM : 0
position_CB : 0
position_CDM : 0
position_CF : 0
position_CM : 0
position_GK : 0
position_LB : 0
position_LM : 0
position_LW : 0
position_LWB : 0
position_RB : 0
position_RM : 0
position_RW : 0
position_RWB : 0
position_ST : 0
Height Category : 0
Weight Category : 1
Wage Category : 0
Value Category : 0
Release Category : 3
```

From the above display, it can be seen that the data has missing values

To handle the missing variables:

- I'll fill the Hits columns with 0 because it is my outcome variable
- since the other columns have little missing values, i'll drop them

In [53]:
```
# In this code, we use the dropna method with the subset parameter:
# This will drop other missing values and fill Hits with 0 when missing values exist

df_cleaned = df_m.dropna(subset=df_m.columns.drop('Hits'), inplace=False)
df_cleaned = df_cleaned.fillna({'Hits': 0.0})
#df_cleaned= df_m['Hits'].fillna(0.0, inplace=True)
```

In [54]:
```
df_cleaned.shape
```

Out[54]:
```
(18883, 67)
```

Checking to see if the nulls are filled

In [55]:

```
# this codes will return the values of nulls present in the data
null_counts = df_cleaned.isnull().sum()
columns_with_null = null_counts[null_counts >= 0]
for column_name, count in columns_with_null.items():
    print(f' {column_name} : {count}')
```

```
Age : 0
Preferred Foot : 0
Attacking : 0
Crossing : 0
Finishing : 0
Heading Accuracy : 0
Short Passing : 0
Volleys : 0
Skill : 0
Dribbling : 0
Curve : 0
FK Accuracy : 0
Long Passing : 0
Ball Control : 0
Movement : 0
Acceleration : 0
Sprint Speed : 0
Agility : 0
Reactions : 0
Balance : 0
Power : 0
Shot Power : 0
Jumping : 0
Stamina : 0
Strength : 0
Long Shots : 0
Mentality : 0
Aggression : 0
Interceptions : 0
Positioning : 0
Vision : 0
Penalties : 0
Composure : 0
Defending : 0
Marking : 0
Standing Tackle : 0
Sliding Tackle : 0
Goalkeeping : 0
GK Diving : 0
GK Handling : 0
GK Kicking : 0
GK Positioning : 0
GK Reflexes : 0
```

```
Total Stats : 0
Base Stats : 0
Hits : 0
Player Status : 0
position_CAM : 0
position_CB : 0
position_CDM : 0
position_CF : 0
position_CM : 0
position_GK : 0
position_LB : 0
position_LM : 0
position_LW : 0
position_LWB : 0
position_RB : 0
position_RM : 0
position_RW : 0
position_RWB : 0
position_ST : 0
Height Category : 0
Weight Category : 0
Wage Category : 0
Value Category : 0
Release Category : 0
```

In [ ]:

```

```

In [ ]:

```

```

Task 2:

*Preprocess the cleaned data from **task 1** above and transform it into a well behaved data.*

Steps to achive the task 2:

- Removing outliers
- Skew the numeric data
- Encode categorical columns to be numeric
- Standardize

- Normalize the data

In [57]:
```
df_cleaned.describe(include = ['object','category']) # this will return the info for categorical columns
```

Out[57]:

	Preferred Foot	Short Passing	Dribbling	Player Status	Height Category	Weight Category	Wage Category	Value Category	Release Category
count	18883	18883	18883	18883	18883	18883	18883	18883	18883
unique	2	153	165	3	6	6	8	4	5
top	Right	62	62	Active	Below Average	Below Average	Very Low	Small	Beginner
freq	14369	631	514	17636	9407	9937	18718	18778	18722

In [58]:
```
df_cleaned[['Short Passing', 'Dribbling']]
```

Out[58]:

	Short Passing	Dribbling
0	91	96
1	82	88
2	43	12
3	94	88
4	87	95
...	...	...
19016	26.0	27
19017	56.0	46
19018	54.0	43
19019	42.0	51
19020	45.0	40

18883 rows × 2 columns

In [59]:
```
df_cleaned[['Short Passing', 'Dribbling']].dtypes
```

Out[59]:
```
Short Passing    object
Dribbling        object
dtype: object
```

In [60]:
```
# convert the data type to  floats
df_cleaned['Short Passing'] = df_cleaned['Short Passing'].apply(lambda x: float(x.replace('_', ''))
                                                                 if isinstance(x, str) and '_' in x else f


df_cleaned['Dribbling'] = df_cleaned['Dribbling'].apply(lambda x: float(x.replace('_', ''))
                                                                 if isinstance(x, str) and '_' in x else f

```

Seperate the data into categorical and continuous to treat outliers

In [61]:
```
# sepearte the data into categorical and numerical

continuous_vars=df_cleaned.select_dtypes(include=['int64', 'float64']).columns
print(continuous_vars)
print('--------------------------------------------------')
categorical_vars=df_cleaned.select_dtypes(include=['object','category']).columns
print(categorical_vars)

```

```
Index(['Age', 'Attacking', 'Crossing', 'Finishing', 'Heading Accuracy',
       'Short Passing', 'Volleys', 'Skill', 'Dribbling', 'Curve',
       'FK Accuracy', 'Long Passing', 'Ball Control', 'Movement',
       'Acceleration', 'Sprint Speed', 'Agility', 'Reactions', 'Balance',
       'Power', 'Shot Power', 'Jumping', 'Stamina', 'Strength', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Penalties', 'Composure', 'Defending', 'Marking', 'Standing Tackle',
       'Sliding Tackle', 'Goalkeeping', 'GK Diving', 'GK Handling',
       'GK Kicking', 'GK Positioning', 'GK Reflexes', 'Total Stats',
       'Base Stats', 'Hits', 'position_CAM', 'position_CB', 'position_CDM',
       'position_CF', 'position_CM', 'position_GK', 'position_LB',
       'position_LM', 'position_LW', 'position_LWB', 'position_RB',
       'position_RM', 'position_RW', 'position_RWB', 'position_ST'],
      dtype='object')
--------------------------------------------------
Index(['Preferred Foot', 'Player Status', 'Height Category', 'Weight Category',
       'Wage Category', 'Value Category', 'Release Category'],
      dtype='object')
```

Inspecting for outliers in our data

In [62]:

```
# inspecting for outliers in our data

def outlier_lims(col):
    q3,q1 = np.percentile(col, [75,25])
    iqr = q3-q1
    upper_lim = q3 + 1.5*iqr
    lower_lim = q1 - 1.5*iqr
    return upper_lim, lower_lim

for col in continuous_vars:
    print("--------------------------------------------------")
    print("Column:", col)

    UL,LL = outlier_lims(df_cleaned[col])
    print("Upper Limit =", UL)
    print("Lower Limit =", LL)

    total_outliers = len(df_cleaned.loc[df_cleaned[col]<LL,col]) + len(df_cleaned.loc[df_cleaned[col]>UL,
    percent = (total_outliers / len(df_cleaned.index) )*100

    print("Percentage of Outliers=", percent)
    print("---------------------------------------------")
```

```
--------------------------------------------------
Column: Age
Upper Limit = 41.0
Lower Limit = 9.0
Percentage of Outliers= 0.04236614944659217
----------------------------------------------
--------------------------------------------------
Column: Attacking
Upper Limit = 409.5
Lower Limit = 109.5
Percentage of Outliers= 10.655086585817932
----------------------------------------------
--------------------------------------------------
Column: Crossing
Upper Limit = 100.5
Lower Limit = 0.5
Percentage of Outliers= 0.0
----------------------------------------------
--------------------------------------------------
```

Removing outliers with percentage equal to or greater than 10

In [63]:

```python
#Removing outliers
df_cleaned.select_dtypes(include=['int64', 'float64']).skew()
#using log transformation for PhysicalHealth and MentalHealth to reduce skewness
df_cleaned[['Attacking','Dribbling','Ball Control','Goalkeeping','GK Diving','GK Handling','GK Positionin
    'Hits']] = np.log1p(df_cleaned[['Attacking','Dribbling','Ball Control','Goalkeeping','GK Diving','GK
                                    'GK Positioning','GK Reflexes','Hits']])
```

In [64]:
```
for col in continuous_vars:
    print("---------------------------------------------------")
    print("Column:", col)

    UL,LL = outlier_lims(df_cleaned[col])
    print("Upper Limit =", UL)
    print("Lower Limit =", LL)

    total_outliers = len(df_cleaned.loc[df_cleaned[col]<LL,col]) + len(df_cleaned.loc[df_cleaned[col]>UL,
    percent = (total_outliers / len(df_cleaned.index) )*100

    print("Percentage of Outliers=", percent)
    print("-----------------------------------------------")
```

```
--------------------------------------------------
Column: Age
Upper Limit = 41.0
Lower Limit = 9.0
Percentage of Outliers= 0.04236614944659217
----------------------------------------------
--------------------------------------------------
Column: Attacking
Upper Limit = 6.131976059073333
Lower Limit = 4.97228919889219
Percentage of Outliers= 11.03638193083726
----------------------------------------------
--------------------------------------------------
Column: Crossing
Upper Limit = 100.5
Lower Limit = 0.5
Percentage of Outliers= 0.0
----------------------------------------------
--------------------------------------------------
```

Encode the categorical values

In [65]:
```
categorical_vars
```

Out[65]:
```
Index(['Preferred Foot', 'Player Status', 'Height Category', 'Weight Category',
       'Wage Category', 'Value Category', 'Release Category'],
      dtype='object')
```

In [66]:
```
## Handling categorical data (label encoding)
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df_cleaned['Preferred Foot'] = encoder.fit_transform(df_cleaned['Preferred Foot'])
df_cleaned['Player Status'] = encoder.fit_transform(df_cleaned['Player Status'])
df_cleaned['Height Category'] = encoder.fit_transform(df_cleaned['Height Category'])
df_cleaned['Weight Category'] = encoder.fit_transform(df_cleaned['Weight Category'])
df_cleaned['Wage Category'] = encoder.fit_transform(df_cleaned['Wage Category'])
df_cleaned['Value Category'] = encoder.fit_transform(df_cleaned['Value Category'])
df_cleaned['Release Category'] = encoder.fit_transform(df_cleaned['Release Category'])
```

In [67]:
```
df_cleaned
```

Out[67]:

	Age	Preferred Foot	Attacking	Crossing	Finishing	Heading Accuracy	Short Passing	Volleys	Skill	Dribbling	...	position_RB	position_RM	posi
0	33	0	6.063785	85.0	95.0	70.0	91.0	88.0	470.0	4.574711	...	0	0	
1	35	1	6.082219	84.0	95.0	90.0	82.0	86.0	414.0	4.488636	...	0	0	
2	27	1	4.564348	13.0	11.0	15.0	43.0	13.0	109.0	2.564949	...	0	0	
3	29	1	6.011267	94.0	82.0	55.0	94.0	82.0	441.0	4.488636	...	0	0	
4	28	1	6.013715	85.0	87.0	62.0	87.0	87.0	448.0	4.564348	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19016	21	1	4.983607	23.0	26.0	43.0	26.0	27.0	142.0	3.332205	...	0	0	
19017	17	1	5.356586	38.0	42.0	40.0	56.0	35.0	219.0	3.850148	...	0	0	
19018	18	1	5.303305	30.0	34.0	43.0	54.0	39.0	207.0	3.784190	...	0	0	
19019	20	1	5.375278	45.0	52.0	34.0	42.0	42.0	194.0	3.951244	...	0	0	
19020	21	0	5.099866	40.0	18.0	40.0	45.0	20.0	171.0	3.713572	...	0	0	

18883 rows × 67 columns

Now the data is ready for standardization

Standardising the dataset

In [68]:
```
# Handling numerical data (standardization)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_cleaned[continuous_vars] = scaler.fit_transform(df_cleaned[continuous_vars])
```

In [69]:
```
1 df_cleaned
```

Out[69]:

	Age	Preferred Foot	Attacking	Crossing	Finishing	Heading Accuracy	Short Passing	Volleys	Skill	Dribbling	...	position_RB	posi
0	1.660279	0	1.434591	1.953274	2.516677	1.046913	2.226256	2.574238	2.722313	1.224706	...	-0.348021	-(
1	2.085223	1	1.477667	1.898052	2.516677	2.203890	1.605698	2.460737	2.009326	1.059417	...	-0.348021	-(
2	0.385449	1	-2.069302	-2.022694	-1.779208	-2.134774	-1.083387	-1.682028	-1.873907	-2.634645	...	-0.348021	-(
3	0.810392	1	1.311867	2.450270	1.851838	0.179180	2.433109	2.233737	2.353088	1.059417	...	-0.348021	-(
4	0.597921	1	1.317587	1.953274	2.107545	0.584122	1.950453	2.517488	2.442211	1.204807	...	-0.348021	-(
...	...	...	...	...	...	...	...	...	...	...	...	...	
19016	-0.889381	1	-1.089577	-1.470476	-1.012086	-0.515006	-2.255552	-0.887525	-1.453754	-1.161282	...	-0.348021	-(
19017	-1.739268	1	-0.217995	-0.642149	-0.193822	-0.688552	-0.187025	-0.433523	-0.473397	-0.166675	...	-0.348021	-(
19018	-1.526797	1	-0.342504	-1.083924	-0.602954	-0.515006	-0.324927	-0.206522	-0.626180	-0.293334	...	-0.348021	-(
19019	-1.101853	1	-0.174316	-0.255597	0.317593	-1.035646	-1.152338	-0.036272	-0.791695	0.027460	...	-0.348021	-(
19020	-0.889381	0	-0.817900	-0.531706	-1.421218	-0.688552	-0.945485	-1.284776	-1.084529	-0.428941	...	-0.348021	-(

18883 rows × 67 columns

In [ ]:
```
1
```

Normalizing the data

In [71]:
```
1 df_cleaned.shape
```

Out[71]: (18883, 67)

In [72]:
```
1 df_cleaned.columns.get_loc('Hits')
```

Out[72]: 45

In [73]:
```python
df_cleaned.columns.get_loc('Release Category')
```

Out[73]:
```
66
```

In [74]:
```python
dfArr = df_cleaned.values

from sklearn.preprocessing import Normalizer

x = dfArr[:, :66]
y = dfArr[:,45]

norm = Normalizer().fit(x)
Normalizeredx = norm.transform(x)

Normalizeredx
```

Out[74]:
```
array([[ 0.09950342,  0.        ,  0.08597755, ...,  0.05993174,
         0.35959042,  0.        ],
       [ 0.14311048,  0.06863079,  0.10141347, ...,  0.        ,
         0.34315394,  0.13726158],
       [ 0.02785219,  0.0722591 , -0.14952592, ...,  0.        ,
         0.28903639,  0.        ],
       ...,
       [-0.15988593,  0.10471986, -0.03586694, ...,  0.10471986,
         0.73303903,  0.31415958],
       [-0.09609259,  0.08720998, -0.01520206, ...,  0.26162994,
         0.61046986,  0.26162994],
       [-0.08236884,  0.        , -0.07574868, ...,  0.09261362,
         0.64829537,  0.27784087]])
```

In [75]:
```
df_cleaned.columns
```

Out[75]:
```
Index(['Age', 'Preferred Foot', 'Attacking', 'Crossing', 'Finishing',
       'Heading Accuracy', 'Short Passing', 'Volleys', 'Skill', 'Dribbling',
       'Curve', 'FK Accuracy', 'Long Passing', 'Ball Control', 'Movement',
       'Acceleration', 'Sprint Speed', 'Agility', 'Reactions', 'Balance',
       'Power', 'Shot Power', 'Jumping', 'Stamina', 'Strength', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Penalties', 'Composure', 'Defending', 'Marking', 'Standing Tackle',
       'Sliding Tackle', 'Goalkeeping', 'GK Diving', 'GK Handling',
       'GK Kicking', 'GK Positioning', 'GK Reflexes', 'Total Stats',
       'Base Stats', 'Hits', 'Player Status', 'position_CAM', 'position_CB',
       'position_CDM', 'position_CF', 'position_CM', 'position_GK',
       'position_LB', 'position_LM', 'position_LW', 'position_LWB',
       'position_RB', 'position_RM', 'position_RW', 'position_RWB',
       'position_ST', 'Height Category', 'Weight Category', 'Wage Category',
       'Value Category', 'Release Category'],
      dtype='object')
```

Creating a new DataFrame with the nornmalized values

In [76]:

```
new_columns = ['Age', 'Preferred Foot', 'Attacking', 'Crossing', 'Finishing','Heading Accuracy',
'Short Passing', 'Volleys', 'Skill', 'Dribbling',
'Curve', 'FK Accuracy', 'Long Passing', 'Ball Control', 'Movement',
'Acceleration', 'Sprint Speed', 'Agility', 'Reactions', 'Balance',
'Power', 'Shot Power', 'Jumping', 'Stamina', 'Strength', 'Long Shots',
'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
'Penalties', 'Composure', 'Defending', 'Marking', 'Standing Tackle',
'Sliding Tackle', 'Goalkeeping', 'GK Diving', 'GK Handling','GK Kicking', 'GK Positioning', 'GK Reflexes'
'position_LB', 'position_LM', 'position_LW', 'position_LWB',
'position_RB', 'position_RM', 'position_RW', 'position_RWB',
'position_ST', 'Height Category', 'Weight Category', 'Wage Category',
'Value Category', 'Release Category']

df_norm = pd.DataFrame(Normalizeredx, columns =new_columns)



df_norm['Hits'] = df_cleaned['Hits']

df_norm

```

Out[76]:

	Age	Preferred Foot	Attacking	Crossing	Finishing	Heading Accuracy	Short Passing	Volleys	Skill	Dribbling	...	position_RM	posi
0	0.099503	0.000000	0.085978	0.117063	0.150829	0.062743	0.133423	0.154279	0.163153	0.073399	...	-0.020857	-(
1	0.143110	0.068631	0.101413	0.130265	0.172722	0.151255	0.110200	0.168882	0.137902	0.072709	...	-0.023885	-(
2	0.027852	0.072259	-0.149526	-0.146158	-0.128564	-0.154257	-0.078285	-0.121542	-0.135407	-0.190377	...	-0.025148	-(
3	0.062437	0.077045	0.101073	0.188781	0.142675	0.013805	0.187459	0.172098	0.181294	0.081623	...	-0.026813	-(
4	0.044726	0.074802	0.098558	0.146109	0.157649	0.043694	0.145898	0.188313	0.182682	0.090122	...	-0.026033	-(
...	...	...	...	...	...	...	...	...	...	...	...	...	
18878	-0.078559	0.088330	-0.096243	-0.129888	-0.089398	-0.045491	-0.199234	-0.078395	-0.128411	-0.102577	...	-0.030741	-(
18879	-0.170822	0.098215	-0.021410	-0.063068	-0.019036	-0.067626	-0.018369	-0.042578	-0.046495	-0.016370	...	-0.034181	-(
18880	-0.159886	0.104720	-0.035867	-0.113508	-0.063141	-0.053931	-0.034026	-0.021627	-0.065573	-0.030718	...	-0.036445	-(
18881	-0.096093	0.087210	-0.015202	-0.022291	0.027697	-0.090319	-0.100495	-0.003163	-0.069044	0.002395	...	-0.030351	-(
18882	-0.082369	0.000000	-0.075749	-0.049243	-0.131624	-0.063769	-0.087565	-0.118988	-0.100442	-0.039726	...	-0.032231	-(

18883 rows × 67 columns

In [77]:
```
#Check the data for missing values

null_counts = df_norm.isna().sum()
columns_with_null = null_counts[null_counts >= 0]
for column_name, count in columns_with_null.items():
    print(f' {column_name} : {count}')
```

```
Age : 0
Preferred Foot : 0
Attacking : 0
Crossing : 0
Finishing : 0
Heading Accuracy : 0
Short Passing : 0
Volleys : 0
Skill : 0
Dribbling : 0
Curve : 0
FK Accuracy : 0
Long Passing : 0
Ball Control : 0
Movement : 0
Acceleration : 0
Sprint Speed : 0
Agility : 0
Reactions : 0
Balance : 0
Power : 0
Shot Power : 0
Jumping : 0
Stamina : 0
Strength : 0
Long Shots : 0
Mentality : 0
Aggression : 0
Interceptions : 0
Positioning : 0
Vision : 0
Penalties : 0
Composure : 0
Defending : 0
Marking : 0
Standing Tackle : 0
Sliding Tackle : 0
Goalkeeping : 0
GK Diving : 0
GK Handling : 0
GK Kicking : 0
GK Positioning : 0
GK Reflexes : 0
```

```
Total Stats : 0
Base Stats : 0
Player Status : 0
position_CAM : 0
position_CB : 0
position_CDM : 0
position_CF : 0
position_CM : 0
position_GK : 0
position_LB : 0
position_LM : 0
position_LW : 0
position_LWB : 0
position_RB : 0
position_RM : 0
position_RW : 0
position_RWB : 0
position_ST : 0
Height Category : 0
Weight Category : 0
Wage Category : 0
Value Category : 0
Release Category : 0
Hits : 138
```

Task 3.

Select input features for an outcome feature of HITS.

Before we select the input features, we ensure that there is no missing value, because we have handled missing values before now.

In [78]:

```
df_norm.dropna(inplace=True)
```

The code below uses Linear Regresiion Elimination Method to select features

for the model

In [85]:
```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression  # Use LinearRegression for regression tasks

X = df_norm.drop('Hits', axis=1)
y = df_norm['Hits']

# Create a linear regression model
model = LinearRegression()  # Use LinearRegression for regression tasks

# Create the RFE object with the linear regression model and the number of features to select
num_features_to_select = 33
rfe = RFE(model, n_features_to_select=num_features_to_select)

# Fit the RFE object to the data to perform feature selection
rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[rfe.support_]

print("Num Features: %d" % rfe.n_features_)
print(f"Selected Features: " )
print(selected_features)
```

```
Num Features: 33
Selected Features:
Index(['Age', 'Crossing', 'Finishing', 'Heading Accuracy', 'Volleys', 'Skill',
       'Dribbling', 'Ball Control', 'Movement', 'Acceleration', 'Sprint Speed',
       'Agility', 'Reactions', 'Balance', 'Power', 'Long Shots', 'Mentality',
       'Aggression', 'Interceptions', 'Positioning', 'Vision', 'Defending',
       'Marking', 'Standing Tackle', 'Sliding Tackle', 'GK Kicking',
       'Total Stats', 'Base Stats', 'Player Status', 'position_GK',
       'position_LB', 'Value Category', 'Release Category'],
      dtype='object')
```

33 features selceted

In [86]:
```
[rfe.ranking_]  # the ranking of all features
```

Out[86]:
```
[array([ 1, 34, 17,  1,  1,  1, 10,  1,  1,  1, 27, 29, 25,  1,  1,  1,  1,
         1,  1,  1,  1,  3,  6,  5,  4,  1,  1,  1,  1,  1,  1,  8, 13,  1,
         1,  1,  1, 31, 11,  7,  1,  9,  2,  1,  1,  1, 15, 26, 22, 16, 32,
         1,  1, 21, 33, 28, 12, 20, 23, 19, 14, 24, 30, 18,  1,  1])]
```

Let's get thescore for the whole 67 features to compare with the 33 features selcted

In [88]:
```
# Generating the score of the model

model.fit(X,y)

score = model.score(X,y)
score
```

Out[88]:
```
0.48869005568500257
```

The Score of the model for the whole 67 features:0.48869005568500257

Creating a new dataframe for the selected features

In [89]:
```
## Creating a new dataframe for the selected features

df_selected = df_norm[['Age', 'Crossing', 'Finishing', 'Heading Accuracy', 'Volleys', 'Skill',
       'Dribbling', 'Ball Control', 'Movement', 'Acceleration', 'Sprint Speed',
       'Agility', 'Reactions', 'Balance', 'Power', 'Long Shots', 'Mentality',
       'Aggression', 'Interceptions', 'Positioning', 'Vision', 'Defending',
       'Marking', 'Standing Tackle', 'Sliding Tackle', 'GK Kicking',
       'Total Stats', 'Base Stats', 'Player Status', 'position_GK',
       'position_LB', 'Value Category', 'Release Category']]
```

In [90]:

```
# Display the new dataframe

df_selected
```

Out[90]:

	Age	Crossing	Finishing	Heading Accuracy	Volleys	Skill	Dribbling	Ball Control	Movement	Acceleration	...	Standing Tackle	S
0	0.099503	0.117063	0.150829	0.062743	0.154279	0.163153	0.073399	0.080941	0.143184	0.107321	...	-0.035538	-0.0
1	0.143110	0.130265	0.172722	0.151255	0.168882	0.137902	0.072709	0.085628	0.139388	0.104460	...	-0.050340	-0.0
2	0.027852	-0.146158	-0.128564	-0.154257	-0.121542	-0.135407	-0.190377	-0.103800	-0.013687	-0.103561	...	-0.120689	-0.0
3	0.062437	0.188781	0.142675	0.013805	0.172098	0.181294	0.081623	0.096127	0.110950	0.065520	...	0.062571	0.0
4	0.044726	0.146109	0.157649	0.043694	0.188313	0.182682	0.090122	0.099130	0.181389	0.149021	...	-0.061873	-0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	
18878	-0.078559	-0.129888	-0.089398	-0.045491	-0.078395	-0.128411	-0.102577	-0.056270	-0.037294	0.021723	...	0.017954	0.0
18879	-0.170822	-0.063068	-0.019036	-0.067626	-0.042578	-0.046495	-0.016370	-0.031223	-0.022121	-0.008829	...	-0.016838	-0.0
18880	-0.159886	-0.113508	-0.063141	-0.053931	-0.021627	-0.065573	-0.030718	-0.038566	-0.051714	-0.037548	...	-0.022858	-0.0
18881	-0.096093	-0.022291	0.027697	-0.090319	-0.003163	-0.069044	0.002395	-0.036604	-0.099286	-0.013697	...	-0.063967	-0.0
18882	-0.082369	-0.049243	-0.131624	-0.063769	-0.118988	-0.100442	-0.039726	-0.081092	-0.063978	-0.026987	...	-0.007202	-0.0

18745 rows × 33 columns

Training the dataset selected:

- this will check the difference between the overall dataset

In [91]:

```
# training the dataset selected,

X1 = df_selected.values

model.fit(X1,y)

score1 = model.score(X1,y)
score1          # Generating the score of the model
```

Out[91]: 0.4805148870194842

In [ ]:

```

```

Select another 35 features the test the performance of the model

In [98]:

```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression  # Use LinearRegression for regression tasks

X = df_norm.drop('Hits', axis=1)
y = df_norm['Hits']

# Create a linear regression model
model = LinearRegression()  # Use LinearRegression for regression tasks

# Create the RFE object with the linear regression model and the number of features to select (e.g., 20)
num_features_to_select = 35
rfe = RFE(model, n_features_to_select=num_features_to_select)

# Fit the RFE object to the data to perform feature selection
rfe.fit(X, y)

# Get the selected features
selected_features = X.columns[rfe.support_]
#num_feature_sel = X.columns[rfe.support_]
# Print the selected features
print("Num Features: %d" % rfe.n_features_)
print(f"Selected Features: " )
print(selected_features)
```

```
Num Features: 35
Selected Features: 
Index(['Age', 'Crossing', 'Finishing', 'Heading Accuracy', 'Volleys', 'Skill',
       'Dribbling', 'Ball Control', 'Movement', 'Acceleration', 'Sprint Speed',
       'Agility', 'Reactions', 'Balance', 'Power', 'Shot Power', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Defending', 'Marking', 'Standing Tackle', 'Sliding Tackle',
       'GK Kicking', 'GK Reflexes', 'Total Stats', 'Base Stats',
       'Player Status', 'position_GK', 'position_LB', 'Value Category',
       'Release Category'],
      dtype='object')
```

Create a new dataset for the 35 selected features

In [99]:
```
dfsub = df_norm[['Age', 'Crossing', 'Finishing', 'Heading Accuracy', 'Volleys', 'Skill',
       'Dribbling', 'Ball Control', 'Movement', 'Acceleration', 'Sprint Speed',
       'Agility', 'Reactions', 'Balance', 'Power', 'Shot Power', 'Long Shots',
       'Mentality', 'Aggression', 'Interceptions', 'Positioning', 'Vision',
       'Defending', 'Marking', 'Standing Tackle', 'Sliding Tackle',
       'GK Kicking', 'GK Reflexes', 'Total Stats', 'Base Stats',
       'Player Status', 'position_GK', 'position_LB', 'Value Category',
       'Release Category']]
```

Training the dataset selected:

- this will check the difference between the overall dataset

In [100]:
```
# Generating the score of the model

X2 = dfsub.values

model.fit(X2,y)

score2 = model.score(X2,y)
score2
```

Out[100]: 0.48093849829445545

In [ ]:
```



```

Remark:

Overall 35 ffeatures will be selected for model predictions

Reasons:

- when 33 features were selected I got a score of 0.4805148870194842
- when I reduced the features to 30, there was degradation in model perfomance.
- when I increase the numobers to 35 the model improved to 0.48093849829445545

Thanks for viewing my Data Wrangling Project on The Muskets Football Team

In [ ]:

```
1
```