College of Professional Studies
Northeastern University Boston

MPS Analytics

Course:

ALY6040 - Data Mining

Assignment:

Group Project Week 6

Submitted on:

August 16, 2022

Submitted to:

Professor : Diana Puerta

Submitted by:

Ankit Mehra,

Hedy Yi Liu,

Hengyu Liu,

Pavan Kumar Bansal

Contents

Introduction

This project is based on the dataset FIFA19 (sourced from Kaggle) which contains information about different players playing in the league with a total of 18207 rows and 89 columns that includes latest edition FIFA 2019 players attributes like Age, Nationality, Overall, Potential, Club, Value, Wage, Preferred Foot, International Reputation, Weak Foot, Skill Moves, Work Rate, 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, Heading, Accuracy, Short Passing, Volleys, Dribbling, Curve, FK Accuracy, Long Passing, Ball Control, Acceleration, Sprint Speed, Agility, Reactions, Balance, Shot Power, Jumping, Stamina, Strength, Long Shots, Aggression, Interceptions, Positioning, Vision, Penalties, Composure, Marking, Standing Tackle, Sliding Tackle, GK Diving, GK Handling, GK Kicking, GK Positioning, GK Reflexes, Release Clause.

1. Variables of interest:

Below are the variables which we will use during our data analysis :

Variable	Definition
ID	unique id for every player
Name	name of the player
Age	age of the player
Photo	URL to the player's photo
Nationality	nationality of the player
Flag	URL to players' country flag

Overall	overall rating
Potential	potential rating
Club	current club
Club Logo	URL to the club logo
Value	current market value
Wage	current wage
Special	special skill rating
Preferred Foot	left/right
International Reputation	rating on a scale of 5
Weak Foot	rating on a scale of 5
Skill Moves	rating on a scale of 5
Work Rate	attack work rate/defense work rate
Body Type	the body type of player
Real Face	true or false
Position	position on the pitch
Jersey Number	jersey number
Joined	joined date
Loaned From	club name if applicable
Contract Valid Until	contract end date
Height	height of the player
Weight	weight of the player
LS	rating on a scale of 100
ST	rating on a scale of 100

RS	rating on a scale of 100
LW	rating on a scale of 100
LF	rating on a scale of 100
CF	rating on a scale of 100
RF	rating on a scale of 100
RW	rating on a scale of 100
LAM	rating on a scale of 100
CAM	rating on a scale of 100
RAM	rating on a scale of 100
LM	rating on a scale of 100
LCM	rating on a scale of 100
CM	rating on a scale of 100
RCM	rating on a scale of 100
RM	rating on a scale of 100
LWB	rating on a scale of 100
LDM	rating on a scale of 100
CDM	rating on a scale of 100
RDM	rating on a scale of 100
RWB	rating on a scale of 100
LB	rating on a scale of 100
LCB	rating on a scale of 100
CB	rating on a scale of 100
RCB	rating on a scale of 100
RB	rating on a scale of 100

Crossing	rating on a scale of 100
Finishing	rating on a scale of 100
Heading Accuracy	rating on a scale of 100
ShortPassing	rating on a scale of 100
Volleys	rating on a scale of 100
Dribbling	rating on a scale of 100
Curve	rating on a scale of 100
FKAccuracy	rating on a scale of 100
LongPassing	rating on a scale of 100
BallControl	rating on a scale of 100
Acceleration	rating on a scale of 100
SprintSpeed	rating on a scale of 100
Agility	rating on a scale of 100
Reactions	rating on a scale of 100
Balance	rating on a scale of 100
ShotPower	rating on a scale of 100
Jumping	rating on a scale of 100
Stamina	rating on a scale of 100
Strength	rating on a scale of 100
LongShots	rating on a scale of 100
Aggression	rating on a scale of 100
Interceptions	rating on a scale of 100
Positioning	rating on a scale of 100
Vision	rating on a scale of 100

Penalties	rating on a scale of 100
Composure	rating on a scale of 100
Marking	on a scale of 100
StandingTackle	rating on a scale of 100
SlidingTackle	rating on a scale of 100
GKDiving	rating on a scale of 100
GKHandling	rating on a scale of 100
GKKicking	rating on a scale of 100
GKPositioning	rating on a scale of 100
GKReflexes	rating on a scale of 100
Release Clause	release clause value

2. Data types :

Below is the list of the data type of the above-mentioned variables :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 89 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                18207 non-null  int64
 1   ID                        18207 non-null  int64
 2   Name                      18207 non-null  object
 3   Age                       18207 non-null  int64
 4   Photo                     18207 non-null  object
 5   Nationality               18207 non-null  object
 6   Flag                      18207 non-null  object
 7   Overall                   18207 non-null  int64
 8   Potential                 18207 non-null  int64
 9   Club                      17966 non-null  object
 10  Club Logo                 18207 non-null  object
 11  Value                     18207 non-null  object
 12  Wage                      18207 non-null  object
 13  Special                   18207 non-null  int64
 14  Preferred Foot            18159 non-null  object
 15  International Reputation  18159 non-null  float64
 16  Weak Foot                 18159 non-null  float64
 17  Skill Moves               18159 non-null  float64
 18  Work Rate                 18159 non-null  object
 19  Body Type                 18159 non-null  object
 20  Real Face                 18159 non-null  object
 21  Position                  18147 non-null  object
 22  Jersey Number             18147 non-null  float64
 23  Joined                    16654 non-null  object
 24  Loaned From               1264 non-null   object
 25  Contract Valid Until      17918 non-null  object
 26  Height                    18159 non-null  object
 27  Weight                    18159 non-null  object
 28  LS                        16122 non-null  object
 29  ST                        16122 non-null  object
 30  RS                        16122 non-null  object
 31  LW                        16122 non-null  object
 32  LF                        16122 non-null  object
 33  CF                        16122 non-null  object
 34  RF                        16122 non-null  object
 35  RW                        16122 non-null  object
 36  LAM                       16122 non-null  object
 37  CAM                       16122 non-null  object
 38  RAM                       16122 non-null  object
 39  LM                        16122 non-null  object
 40  LCM                       16122 non-null  object
 41  CM                        16122 non-null  object
 42  RCM                       16122 non-null  object
 43  RM                        16122 non-null  object
 44  LWB                       16122 non-null  object
 45  LDM                       16122 non-null  object
 46  CDM                       16122 non-null  object
 47  RDM                       16122 non-null  object
 48  RWB                       16122 non-null  object
 49  LB                        16122 non-null  object
 50  LCB                       16122 non-null  object
 51  CB                        16122 non-null  object
 52  RCB                       16122 non-null  object
 53  RB                        16122 non-null  object
 54  Crossing                  18159 non-null  float64
 55  Finishing                 18159 non-null  float64
 56  HeadingAccuracy           18159 non-null  float64
 57  ShortPassing              18159 non-null  float64
 58  Volleys                   18159 non-null  float64
 59  Dribbling                 18159 non-null  float64
 60  Curve                     18159 non-null  float64
 61  FKAccuracy                18159 non-null  float64
 62  LongPassing               18159 non-null  float64
 63  BallControl               18159 non-null  float64
 64  Acceleration              18159 non-null  float64
 65  SprintSpeed               18159 non-null  float64
 66  Agility                   18159 non-null  float64
```

```
 67  Reactions                 18159 non-null  float64
 68  Balance                   18159 non-null  float64
 69  ShotPower                 18159 non-null  float64
 70  Jumping                   18159 non-null  float64
 71  Stamina                   18159 non-null  float64
 72  Strength                  18159 non-null  float64
 73  LongShots                 18159 non-null  float64
 74  Aggression                18159 non-null  float64
 75  Interceptions             18159 non-null  float64
 76  Positioning               18159 non-null  float64
 77  Vision                    18159 non-null  float64
 78  Penalties                 18159 non-null  float64
 79  Composure                 18159 non-null  float64
 80  Marking                   18159 non-null  float64
 81  StandingTackle            18159 non-null  float64
 82  SlidingTackle             18159 non-null  float64
 83  GKDiving                  18159 non-null  float64
 84  GKHandling                18159 non-null  float64
 85  GKKicking                 18159 non-null  float64
 86  GKPositioning             18159 non-null  float64
 87  GKReflexes                18159 non-null  float64
 88  Release Clause            16643 non-null  object
dtypes: float64(38), int64(6), object(45)
memory usage: 12.4+ MB
```

Console Output

3. Objective :

From the given dataset based on the various attributes, we want to find out the most significant columns to determine the release clause value of all the players playing in the FIFA tournament. Release clause values are the amount of money each player is paid to play the tournament depending on his performance in previous games, wins, best positions, etc.

Our objectives in the dataset are as follows :

1)Use different classification models to predict the release clause class of the players and check class imbalance. We want to classify if the players belong to the low- or high-income class in terms of the release clause.

2) Build regression models to predict the release clause value of the players.

3) Data cleaning and Exploratory data analysis.

4) Hypothesis testing to find out the significant columns for the target variable.

4. Methods that are going to be used in this data analysis:

There are certain types of data analysis, which we will be including to describe the nature of this dataset, such as

- Data Preparation and Cleaning
- Creating visualizations to represent the data; including histograms, boxplots, scatterplots
- descriptive statistics tables
- Hypothesis testing (One-sample, Two sample t-test)
- Classification & regression modelling
- Word Cloud

5. Importing the libraries :

We are working using the python code, so first, we imported the libraries such as pandas, numpy, statistics, matplotlib, warning, and seaborn, as follows :

```
import numpy as np
import pandas as pd
import statistics as stat
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
warnings.filterwarnings("ignore")
```

Script

Interpretation:

- Pandas library is used for data analysis and manipulation
- Matplotlib and seaborn are used for plotting the graphs

- Numpy is to provide support while working with multidimensional data
- Seaborn is for plotting the data
- Warning library is to ignore the warning that is not critical

6. Importing the FIFA19 dataset CSV and named it <data>:

We used the read_csv command to import the FIFA19 dataset, as follows :

```
In [3]: data
```

Out[3]:

	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	...	Composure	Marking	StandingTackle	Sl
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	FC Barcelona	...	96.0	33.0	28.0	
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Juventus	...	95.0	28.0	31.0	
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris Saint-Germain	...	94.0	27.0	24.0	
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Manchester United	...	68.0	15.0	21.0	
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Manchester City	...	88.0	68.0	58.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18202	18202	238813	J. Lundstram	19	https://cdn.sofifa.org/players/4/19/238813.png	England	https://cdn.sofifa.org/flags/14.png	47	65	Crewe Alexandra	...	45.0	40.0	48.0	
18203	18203	243165	N. Christoffersson	19	https://cdn.sofifa.org/players/4/19/243165.png	Sweden	https://cdn.sofifa.org/flags/46.png	47	63	Trelleborgs FF	...	42.0	22.0	15.0	
18204	18204	241638	B. Worman	16	https://cdn.sofifa.org/players/4/19/241638.png	England	https://cdn.sofifa.org/flags/14.png	47	67	Cambridge United	...	41.0	32.0	13.0	
18205	18205	246268	D. Walker-Rice	17	https://cdn.sofifa.org/players/4/19/246268.png	England	https://cdn.sofifa.org/flags/14.png	47	66	Tranmere Rovers	...	46.0	20.0	25.0	
18206	18206	246269	G. Nugent	16	https://cdn.sofifa.org/players/4/19/246269.png	England	https://cdn.sofifa.org/flags/14.png	46	66	Tranmere Rovers	...	43.0	40.0	43.0	

18207 rows × 89 columns

Script

Using the info command we can get a glimpse of the data, which we mentioned in point 2 above in the report.

As per the above outputs, below is the interpretation for structure:

- There is a total of 18207 records in this dataset with 89 variables
- There are columns with data types such as:
  - Float64
  - Object /string
  - Int64

- There is an unnamed column present that is duplicate to ID and hence has no use, so we will eliminate it while preparing the data for analysis
- There are photo URL columns such as Photo, Flag, and Club logo which have no meaning in data analysis, so we will drop these columns as well

7. Data Preparation and Cleaning:

We have followed the below approach for cleaning and preparing the dataset before the analysis :

a. Dropping the columns which are not useful:

As mentioned above, we dropped the following columns :

- Photo
- Flag
- Club Logo
- Unnamed

And then checked the shape of the dataset, which remained as 18207 rows with 85 columns now :

```
In [5]: # renaming columns

In [6]: data.rename(columns = {'Unnamed: 0':'Duplicate'}, inplace = True)

In [7]: # dropping unuseful columns

In [8]: data.drop(['Photo','Flag','Club Logo','Duplicate'],axis=1,inplace=True)
        data.shape

Out[8]: (18207, 85)
```

Script

b. Data manipulation :

There are currency columns such as: “value”,” Wage,” and” Release clause,” which have currency count as M or B as a suffix, and “€” as a prefix. So, for data analysis, we will convert these columns into the numeric type and will remove the suffix using python:

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

	Value	Wage	Release Clause
0	€110.5M	€565K	€226.5M
1	€77M	€405K	€127.1M
2	€118.5M	€290K	€228.1M
3	€72M	€260K	€138.6M
4	€102M	€355K	€196.4M
...	...	...	...
18202	€60K	€1K	€143K
18203	€60K	€1K	€113K
18204	€60K	€1K	€165K
18205	€60K	€1K	€143K
18206	€60K	€1K	€165K

18207 rows × 3 columns

```
def Value_float(Value):
    if isinstance(Value,str):
        out = Value.replace('€', '')
        if 'M' in out:
            out = float(out.replace('M', ''))*1000000
        elif 'K' in Value:
            out = float(out.replace('K', ''))*1000
        return float(out)
data['Value'] = data['Value'].apply(Value_float)
data['Wage'] = data['Wage'].apply(Value_float)
data['Release Clause'] = data['Release Clause'].apply(Value_float)
```

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

	Value	Wage	Release Clause
0	110500000.0	565000.0	226500000.0
1	77000000.0	405000.0	127100000.0
2	118500000.0	290000.0	228100000.0
3	72000000.0	260000.0	138600000.0
4	102000000.0	355000.0	196400000.0
...	...	...	...
18202	60000.0	1000.0	143000.0
18203	60000.0	1000.0	113000.0
18204	60000.0	1000.0	165000.0

Script

c. Building the target column based on the release column values:

As we mentioned in the objective, here we created a class column “Release clause class (0/1)”, where any player with a release clause value less than a million will be in class 0, and above a million will be in class 1, which are representing the lower and higher class respectively,

```
# target column - release clause class
def income(x):
    if x<1000000:
        return 'low_income_class'
    elif x>=1000000:
        return 'high_income_class'
data["Release_Clause_Class"] = data["Release Clause"].apply(income)
data
```

	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	...	Marking	StandingTackle	SlidingTackle	GKDi
0	158023	L. Messi	31	Argentina	94	94	FC Barcelona	110500000.0	565000.0	2202	...	33.0	28.0	26.0	
1	20801	Cristiano Ronaldo	33	Portugal	94	94	Juventus	77000000.0	405000.0	2228	...	28.0	31.0	23.0	
2	190871	Neymar Jr	26	Brazil	92	93	Paris Saint-Germain	118500000.0	290000.0	2143	...	27.0	24.0	33.0	
3	193080	De Gea	27	Spain	91	93	Manchester United	72000000.0	260000.0	1471	...	15.0	21.0	13.0	
4	192985	K. De Bruyne	27	Belgium	91	92	Manchester City	102000000.0	355000.0	2281	...	68.0	58.0	51.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18201	243413	D. Walsh	18	Republic of Ireland	47	68	Waterford FC	60000.0	1000.0	1098	...	44.0	47.0	53.0	
18202	238813	J. Lundstram	19	England	47	65	Crewe Alexandra	60000.0	1000.0	1307	...	40.0	48.0	47.0	
18203	243165	N. Christoffersson	19	Sweden	47	63	Trelleborgs FF	60000.0	1000.0	1098	...	22.0	15.0	19.0	
18204	241638	B. Worman	16	England	47	67	Cambridge United	60000.0	1000.0	1189	...	32.0	13.0	11.0	
18205	246268	D. Walker-Rice	17	England	47	66	Tranmere Rovers	60000.0	1000.0	1228	...	20.0	25.0	27.0	

18206 rows × 86 columns

Script

```
data["Release_Clause_Class"] = data["Release_Clause_Class"].map({"low_income_class":0,
                                                                 "high_income_class":1})
```

Script

```
data["Release Clause"].min()
```

```
13000.0
```

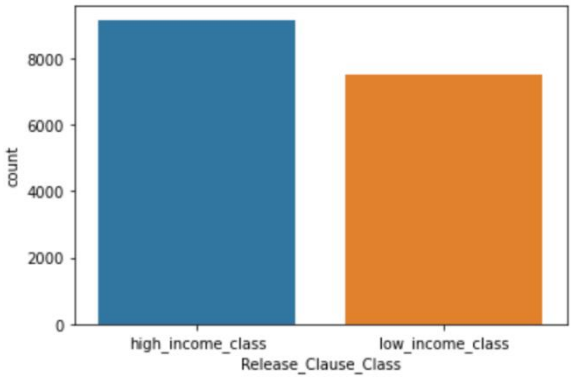
```
data["Release Clause"].max()
```

```
228100000.0
```

```
data["Release_Clause_Class"].value_counts()
```

```
high_income_class    9150
low_income_class     7492
Name: Release_Clause_Class, dtype: int64
```

```
sns.countplot(x="Release_Clause_Class",data=data)
plt.show()
```



Script

- The minimum Release clause is:13000
- The maximum Release clause is:228100000
- And the count of each class is :
    - High class: 9150
    - Low class: 7492

d. Extract the year from the "Joined" column :

We checked for the null values in Joined column and then extracted the year value from it as follows :

```
# converting columns
```

```
data['Joined'].isna().sum()
```

```
1553
```

```
data['Joined']=data['Joined'].fillna('0')
```

```
data['Joined'].isna().sum()
```

```
0
```

```
data['Joined'].head()
```

```
0     Jul 1, 2004
1    Jul 10, 2018
2     Aug 3, 2017
3     Jul 1, 2011
4    Aug 30, 2015
Name: Joined, dtype: object
```

```
def str4(x):
    if isinstance(x,str):
        return int(x[-4:])
data['Joined'] = data['Joined'].apply(str4)
```

Script

Then the updated joined column would be :

```
data['Joined'].head()
```

```
0    2004
1    2018
2    2017
3    2011
4    2015
Name: Joined, dtype: int64
```

Script

e. Convert the "Contract valid until" column into a date column :

The "contract valid until" column stores the date-type data but the column is a string type, so we converted the data into date format :

```
data['Contract Valid Until'].head()
```

```
0    2021
1    2022
2    2022
3    2020
4    2023
Name: Contract Valid Until, dtype: object
```

```
data['Contract Valid Until']= pd.to_datetime(data['Contract Valid Until'])
```

```
data['Contract Valid Until'].head()
```

```
0   2021-01-01
1   2022-01-01
2   2022-01-01
3   2020-01-01
4   2023-01-01
Name: Contract Valid Until, dtype: datetime64[ns]
```

Script

f. Convert height and weight columns in the numeric format :

Next, we converted the height and weight column, into the respective unit of feet and lbs :

```
In [31]: data['Height'].head()

Out[31]: 0     5'7
         1     6'2
         2     5'9
         3     6'4
         4    5'11
         Name: Height, dtype: object

In [32]: def feet(x):
             l=[]
             if isinstance(x,str):
                 l = x.split("'")
                 i = int(l[0])
                 j = int(l[1])
                 f = ((i*12)+j)/12
                 return f
         data['Height'] = data['Height'].apply(feet

In [33]: data['Height'].head()

Out[33]: 0    5.583333
         1    6.166667
         2    5.750000
         3    6.333333
         4    5.916667
         Name: Height, dtype: float64
```

Script

```
[4]: data['Weight'].head()

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

[5]: def llbs(x):
         l=[]
         if isinstance(x,str):
             l = x.split("lbs")
             i = l[0]
             return float(i)
     data['Weight'] = data['Weight'].apply(llbs)

[6]: data['Weight'].head()

[6]: 0    159.0
     1    183.0
     2    150.0
     3    168.0
     4    154.0
     Name: Weight, dtype: float64
```

Script

g. Imputing missing values :

Using the mean value, we filled the NA values of the below columns :

```
[37]: # filling null values with mean

[38]: data['Weight'].fillna(data['Weight'].mean(), inplace = True)
      data['ShortPassing'].fillna(data['ShortPassing'].mean(), inplace = True)
      data['Volleys'].fillna(data['Volleys'].mean(), inplace = True)
      data['Dribbling'].fillna(data['Dribbling'].mean(), inplace = True)
      data['Curve'].fillna(data['Curve'].mean(), inplace = True)
      data['FKAccuracy'].fillna(data['FKAccuracy'].mean(), inplace = True)
      data['LongPassing'].fillna(data['LongPassing'].mean(), inplace = True)
      data['BallControl'].fillna(data['BallControl'].mean(), inplace = True)
      data['HeadingAccuracy'].fillna(data['HeadingAccuracy'].mean(), inplace = True)
      data['Finishing'].fillna(data['Finishing'].mean(), inplace = True)
      data['Crossing'].fillna(data['Crossing'].mean(), inplace = True)
      data['Height'].fillna(data['Height'].mean(), inplace = True)
      data['Joined'].fillna(data['Joined'].mean(), inplace = True)
```

Script

8. Descriptive statistics:

a. Positions of the players :

```
data['Position'] = data['Position'].replace(np.nan, 0)
pos = list(data['Position'].unique())
print('Total number of unique positions:', len(pos)); print()
print('Positions:', pos)
```

```
Total number of unique positions: 28

Positions: ['RF', 'ST', 'LW', 'GK', 'RCM', 'LF', 'RS', 'RCB', 'LCM', 'CB', 'LDM', 'CAM', 'CDM', 'LS', 'LCB', 'RM', 'LAM', 'LM', 'LB', 'RDM', 'RW', 'CM', 'RB', 'RAM', 'CF', 'RWB', 'LWB', 0]
```

```
positiongroup = data.groupby(data['Position'])
tables = list()
for i in pos:
    a = positiongroup.get_group(i).sort_values('Overall',ascending=0).head(5)
    tables.append(a)
print('Total no. of tables:',len(tables))
```

```
Total no. of tables: 28
```

Script

We can observe that unique positions and the total number of tables are total of 28 which means all the players playing in the FIFA tournament belong to 28 unique tables.

b. Repetition of players :

```
idlist = list()
for i in range(0,len(pos)):
    a = tables[i]['ID']
    idlist.append(set(a.values))
```

```
ComIDs = []; x=0
while x<28:
    l = []; y=27
    while y>=0:
        if x != y:
            set1 = idlist[x].intersection(idlist[y])
            l.extend(list(set1))
            l = list(set(l))
        y -= 1
    ComIDs.extend(l)
    ComIDs = list(set(ComIDs))
    x += 1

if len(ComIDs)==0:
    print('There are no players appearing in more than one table.')
else:
    print('Players appearing in more than one table are:{}'.format(df1[df1['ID'] in ComIDs]))
```

```
There are no players appearing in more than one table.
```

Script

We have checked if the name of the players appearing in the 28 tables is repeating which means if their name is showing in more than one table and from the output, we can observe that no player is appearing in more than one table.

c. The average wage of the top 5 players based on their position:

```
avgwage = list()
for i in range(0,len(pos)):
    m = tables[i]['Wage'].mean()
    avgwage.append(m)
    print('Average wage for the top 5 in {}: {}'.format(pos[i],avgwage[i]))
```

```
Average wage for the top 5 in RF: 148000.0
Average wage for the top 5 in ST: 294000.0
Average wage for the top 5 in LW: 261000.0
Average wage for the top 5 in GK: 192800.0
Average wage for the top 5 in RCM: 240800.0
Average wage for the top 5 in LF: 121200.0
Average wage for the top 5 in RS: 132200.0
Average wage for the top 5 in RCB: 231000.0
Average wage for the top 5 in LCM: 184400.0
Average wage for the top 5 in CB: 139600.0
```

Script

From the output, we can observe that the average wage for the top 5 players is the highest in the ST table with an average of 294000.

d. Top 20 players ranked by overall score:

```
top = data[data['Contract Valid Until']=='2020-01-01'].sort_values('Overall',ascending=0).head(20)
top
```

	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	...	Marking	StandingTackle
3	193080	De Gea	27	Spain	91	93	Manchester United	72000000.0	260000.0	1471	...	15.0	21.0
6	177003	L. Modrić	32	Croatia	91	91	Real Madrid	67000000.0	420000.0	2280	...	60.0	76.0
8	155862	Sergio Ramos	32	Spain	91	91	Real Madrid	51000000.0	380000.0	2201	...	87.0	92.0
5	183277	E. Hazard	27	Belgium	91	91	Chelsea	93000000.0	340000.0	2142	...	34.0	27.0
13	168542	David Silva	32	Spain	90	90	Manchester City	60000000.0	285000.0	2115	...	59.0	53.0
21	179813	E. Cavani	31	Uruguay	89	89	Paris Saint-Germain	60000000.0	200000.0	2161	...	52.0	45.0
24	138956	G. Chiellini	33	Italy	89	89	Juventus	27000000.0	215000.0	1841	...	93.0	93.0
31	190460	C. Eriksen	26	Denmark	88	91	Tottenham Hotspur	73500000.0	205000.0	2117	...	59.0	57.0
39	164240	Thiago Silva	33	Brazil	88	88	Paris Saint-Germain	24000000.0	165000.0	2077	...	88.0	89.0
50	175943	D. Mertens	31	Belgium	87	87	Napoli	45000000.0	135000.0	2043	...	25.0	40.0
52	171877	M. Hamšík	30	Slovakia	87	87	Napoli	46500000.0	125000.0	2188	...	75.0	73.0
49	189332	Jordi Alba	29	Spain	87	87	FC Barcelona	38000000.0	250000.0	2230	...	72.0	84.0
46	193041	K. Navas	31	Costa Rica	87	87	Real Madrid	30500000.0	195000.0	1345	...	28.0	14.0
64	191043	Alex Sandro	27	Brazil	86	86	Juventus	36500000.0	160000.0	2198	...	81.0	84.0

71	184087	T. Alderweireld	29	Belgium	86	87	Tottenham Hotspur	39000000.0	150000.0	2047	...	90.0	91.0
73	177509	M. Benatia	31	Morocco	86	86	Juventus	30000000.0	160000.0	1803	...	89.0	87.0
75	135507	Fernandinho	33	Brazil	86	86	Manchester City	18000000.0	185000.0	2183	...	85.0	85.0
102	171919	Naldo	35	Brazil	85	85	FC Schalke 04	9000000.0	38000.0	1959	...	86.0	88.0
104	168609	Miranda	33	Brazil	85	85	Inter	15500000.0	96000.0	1879	...	90.0	90.0
103	170890	B. Matuidi	31	France	85	85	Juventus	26000000.0	145000.0	2196	...	85.0	84.0

20 rows × 86 columns

Script

This table shows the top 20 players ranked by the overall score with their contract expiring in 2020 and we can observe from the results that the top 4 players have the same overall score of 91.

e. Averages for top 20 players :

```
#average wage
```

```
avg = top['Wage'].mean(); print('Average wage for this set of players is:', avg)
```

```
Average wage for this set of players is: 205450.0
```

```
#average value
```

```
avg1 = top['Value'].mean(); print('Average Value for this set of players is:', avg1)
```

```
Average Value for this set of players is: 43075000.0
```

```
#average release clause
```

```
avg2 = top['Release Clause'].mean(); print('Average Release Clause for this set of players is:', avg2)
```

```
Average Release Clause for this set of players is: 80195000.0
```

```
# percentage increase from value to release clause
```

```
x=avg2/avg1*100
x=x-100
print("on an average top 20 players are paid", round(x,2), "percent more than the actual value")
```

```
on an average top 20 players are paid 86.18 percent more than the actual value
```

```
# average age
```

```
age = top['Age'].mean(); print('Average age is:', age)
```

```
Average age is: 30.65
```

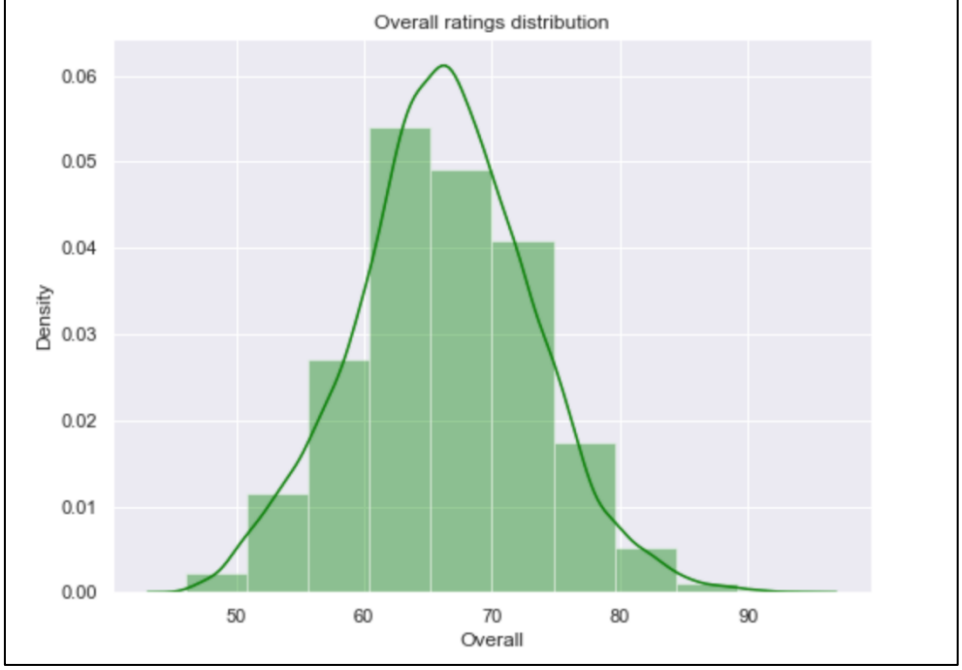
Script

From the output we can observe that for all the top 20 players ranked by overall score their average wage is 205450, the average value is 43075000, the average release clause is 80195000, the average age is 30 years, and based on the release clause on an average the players are paid 86.18% more than their actual value during the FIFA tournament.

9. Data Visualizations:

a. Overall Rating Distribution:

```
sns.set(rc={"figure.figsize": (8, 6)})
plt.title("Overall ratings distribution")
ax = sns.distplot(data["Overall"],color="green",bins=10,kde=True)
plt.show()
```



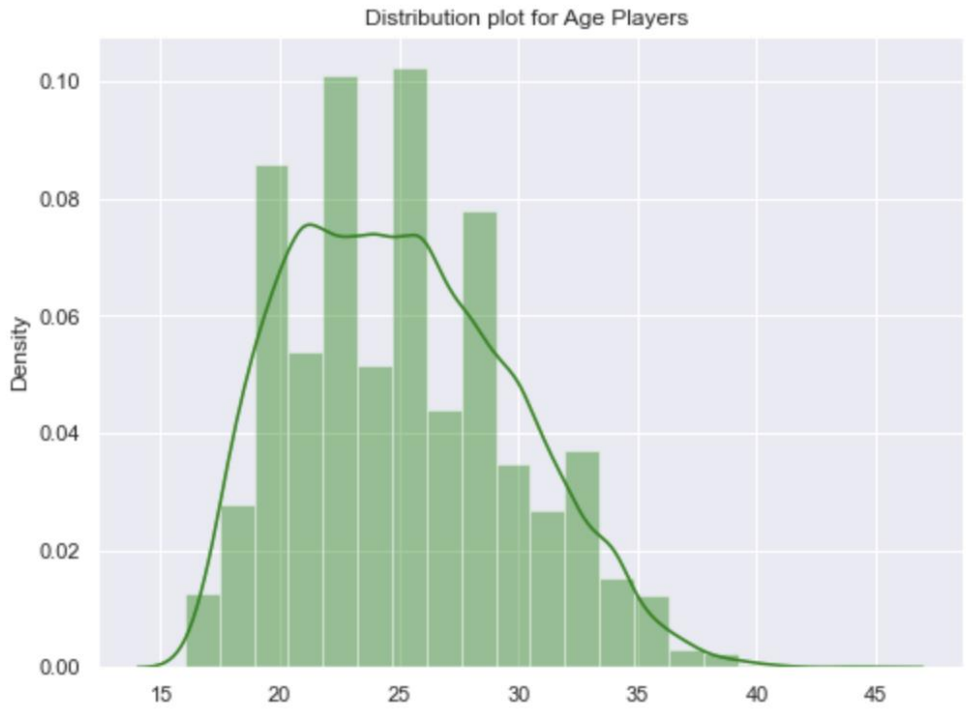
Script

We can observe that most of the players have mean overall ratings of 65 as most of the values are concentrated towards the mean and the graph follows a normal distribution by forming a bell-shaped curve.

b. Age Distribution:

```
sns.set(rc={"figure.figsize": (8, 6)})
plt.title("Distribution plot for Age Players ")
x=sns.distplot(x=data["Age"],color="green",bins=20,kde=True)
print("age of oldest player is : ", data["Age"].max())
print("age of youngest player is : ", data["Age"].min())
print("average age of players is : ", data["Age"].mean())
plt.show()
```

```
age of oldest player is :  45
age of youngest player is :  16
average age of players is :  25.122205745043114
```

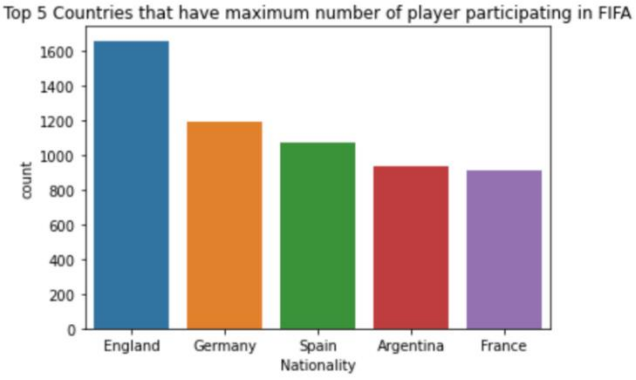


Script

We can observe that the distribution for the age of players is not completely normally distributed as it is a bit right skewed and on average the players participating in the FIFA tournament are of 25 years of age.

c. Nationality:

```
sns.countplot("Nationality", data=data, order=data["Nationality"].value_counts().index[:5])
plt.title("Top 5 Countries that have maximum number of player participating in FIFA")
plt.show()
```



Script

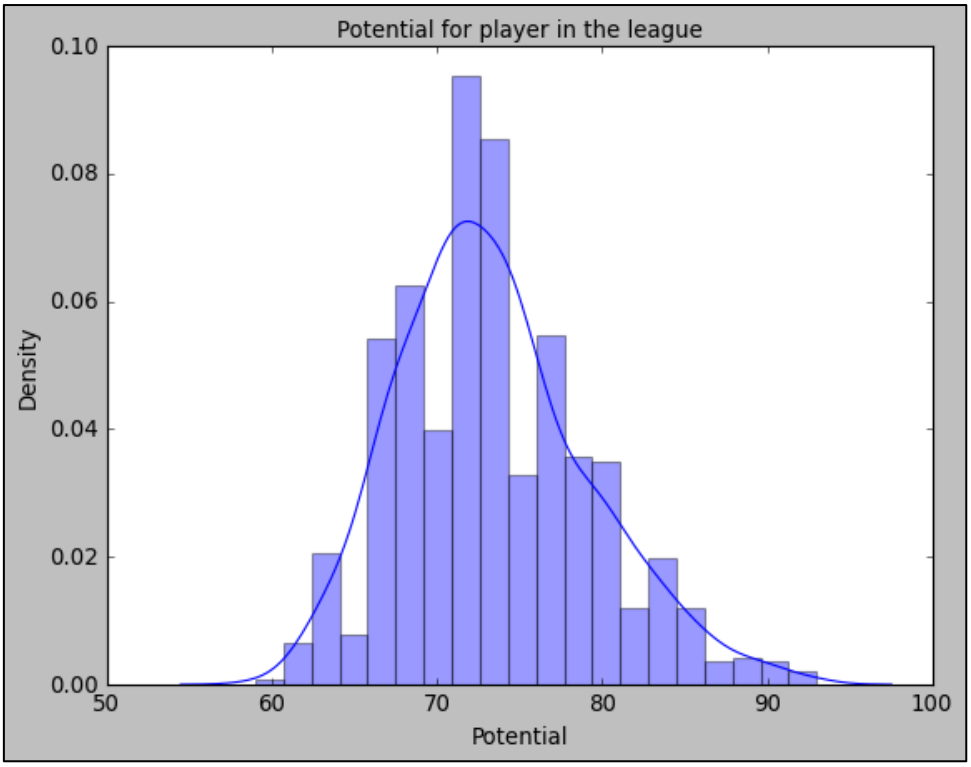
```
data["Nationality"].value_counts()
```

```
England      1662
Germany      1198
Spain        1072
Argentina     937
France        914
```

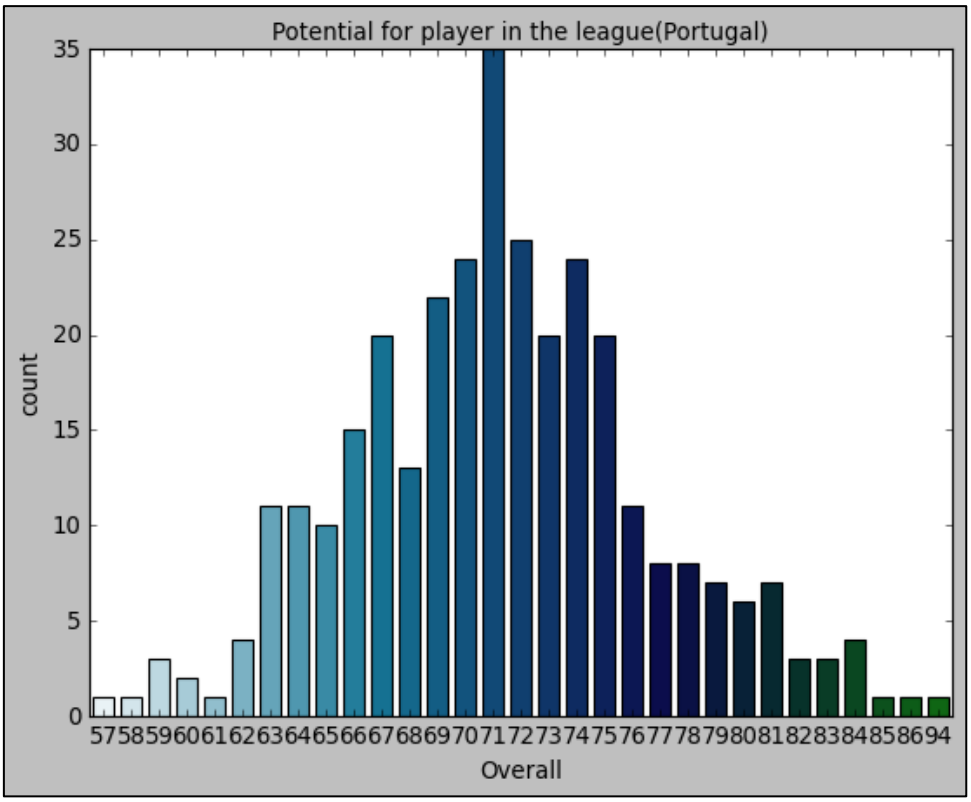
Script

From the count plot, we can observe that England has the maximum number of players that participate in the FIFA tournament with a total of 1662 and France has the least number of participants among the top 5 countries with a total of 914 players participating in the FIFA tournament.

d. Potential for players histogram:


Potential for player in the league
Density
0.10
0.08
0.06
0.04
0.02
0.00
50
60
70
80
90
100
Potential

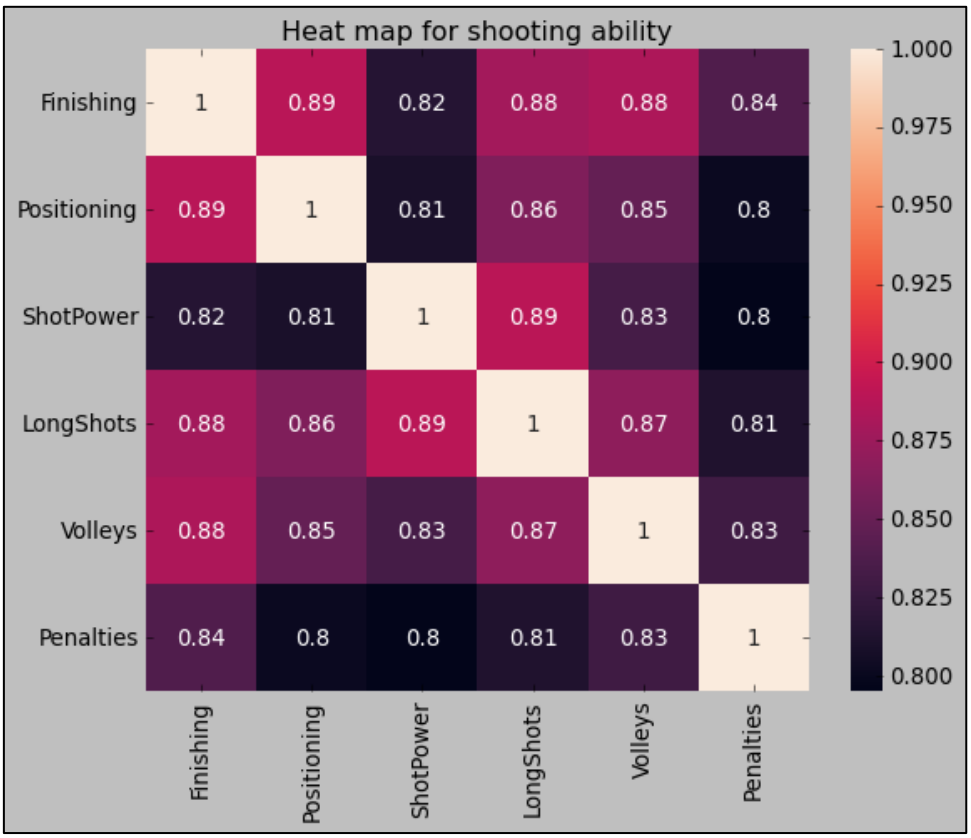
plot


Potential for player in the league(Portugal)
count
35
30
25
20
15
10
5
0
Overall

plot

From the histogram plot above, we can see that the histogram appears at the highest frequency at a rating of around 72. The same frequency pattern shows the Potential for players in the league in Portugal. The extremely talented players are only a few, which correspond to reality.

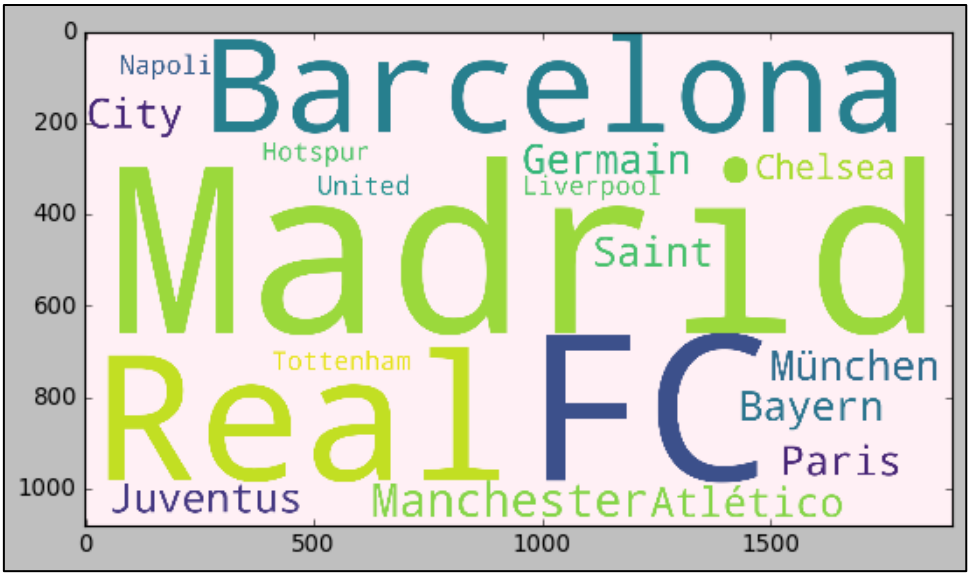
e. Correlation factors with shooting ability:



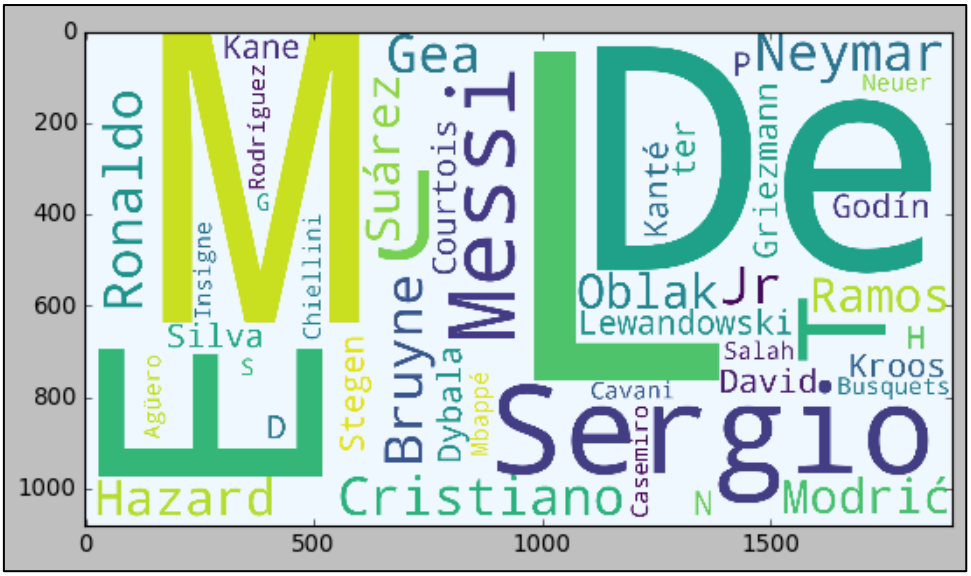
plot

The heat map gives a very intuitive view of what are the most crucial factors that affected the shooting ability. According to the color scale, the deeper color indicated the correlation is less and the sallow color indicated to have a strong correlation with the finishing shooting ability. Overall, in the graph, all these are critical factors, but the most key factor shown is "positioning" and the less crucial factor is "shot power."

f. Word-Cloud Visualization:



plot

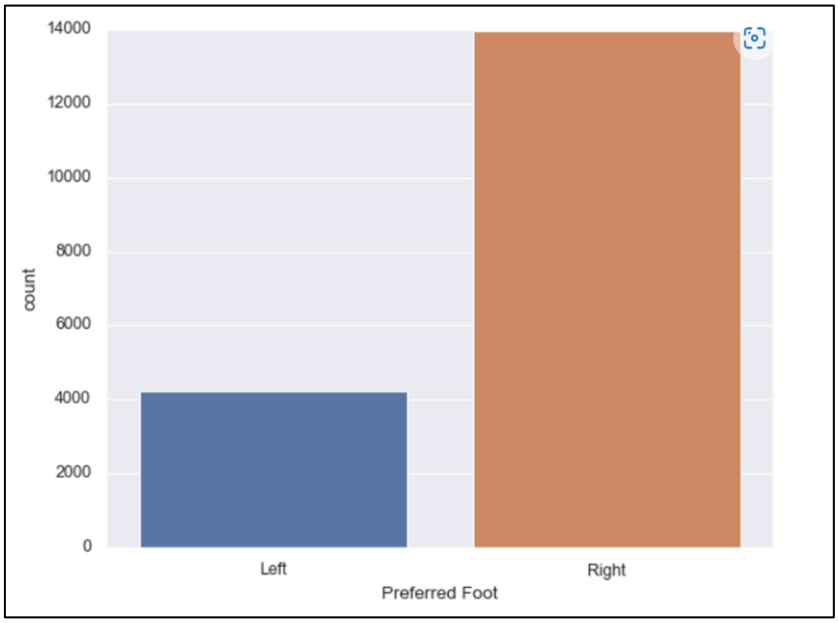


plot

Another visualization we made is the word cloud. Word cloud is an immensely popular method to show what is the hottest topic through the dataset. From the poster, we can see the popular player like L. Messi, Cristiano Ronaldo, Neymar Jr, and De Gea shown in our Word cloud. FC

Barcelona, Juventus, Manchester United, Manchester City, Chelsea, Real Madrid, and FC Barcelona are the most popular team in FIFA 2019.

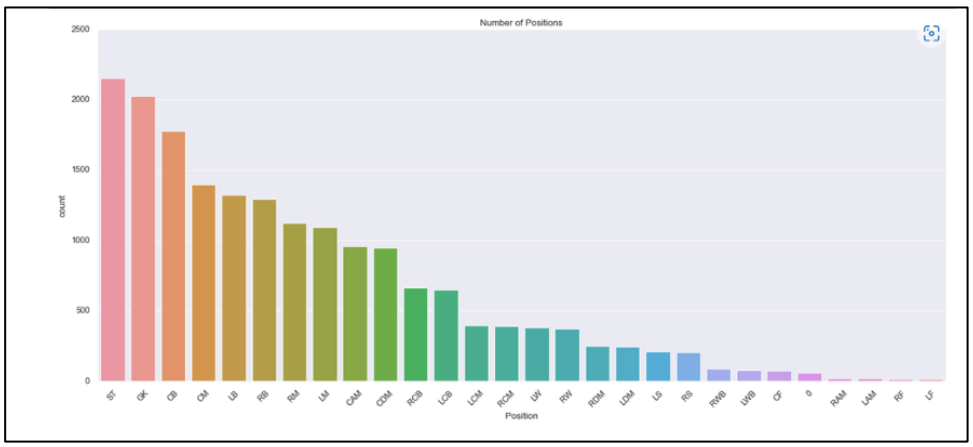
g. Shooting habits



plot

As you can see from the picture, the preferred foot to use is the right foot. This was a much more popular foot than the left and was three times more popular.

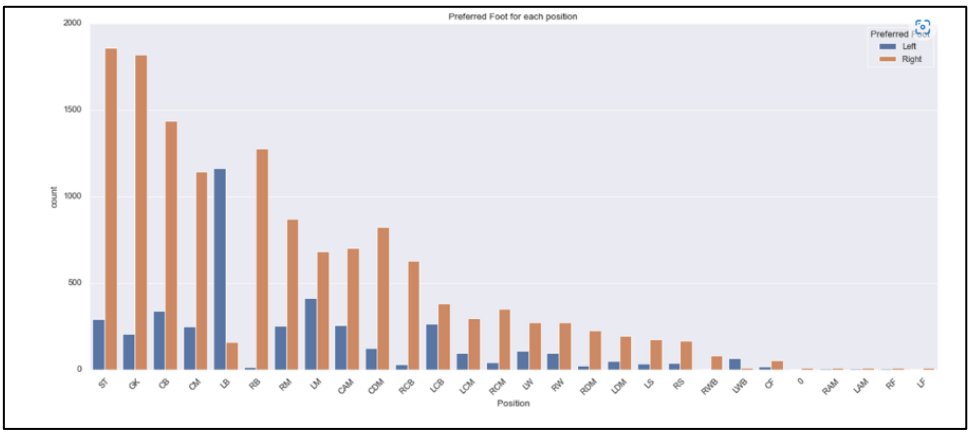
h. Number of Positions:



plot

In terms of positions used by FIFA players, the most popular position is ST, followed by GK. For the use of player positions, people's preferences are very concentrated, with the most popular three positions being used for more than one-fifth of the total.

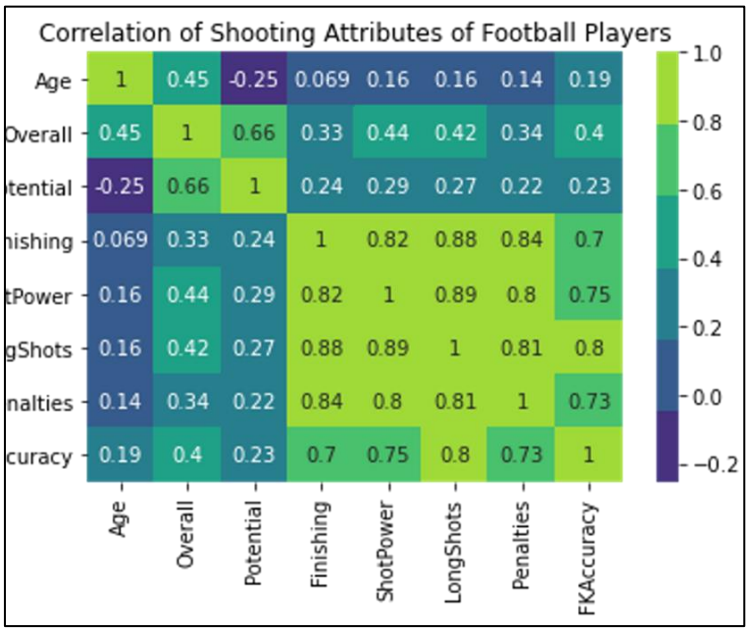
i. Preferred for each position:



plot

In all positions, people are right-handed, meaning they play with their right foot, but at LB, people prefer to play with their left foot.

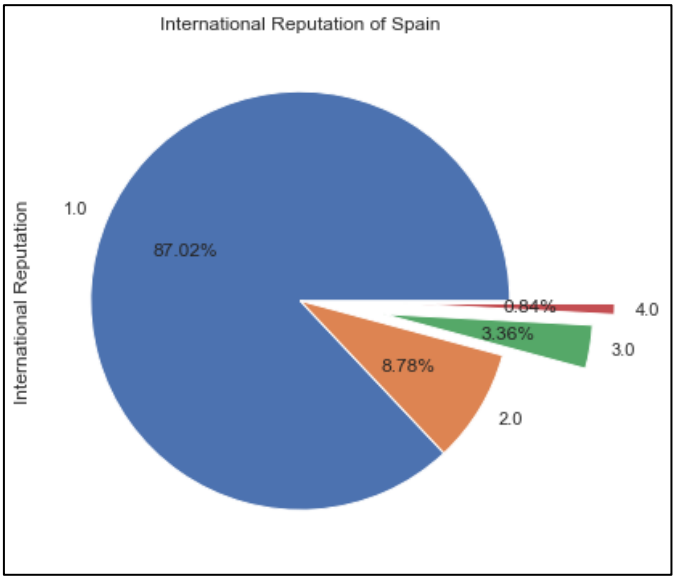
j. Relationship between players' shooting and other attributes:



plot

This is the relationship between players' shooting and other attributes. From this relationship, we can see that some variables are strongly correlated, while some variables are not. Among them, the most relevant to shot accuracy are shot completion ability, shot power, and long shot ability.

k. The international reputation of players of Spain:

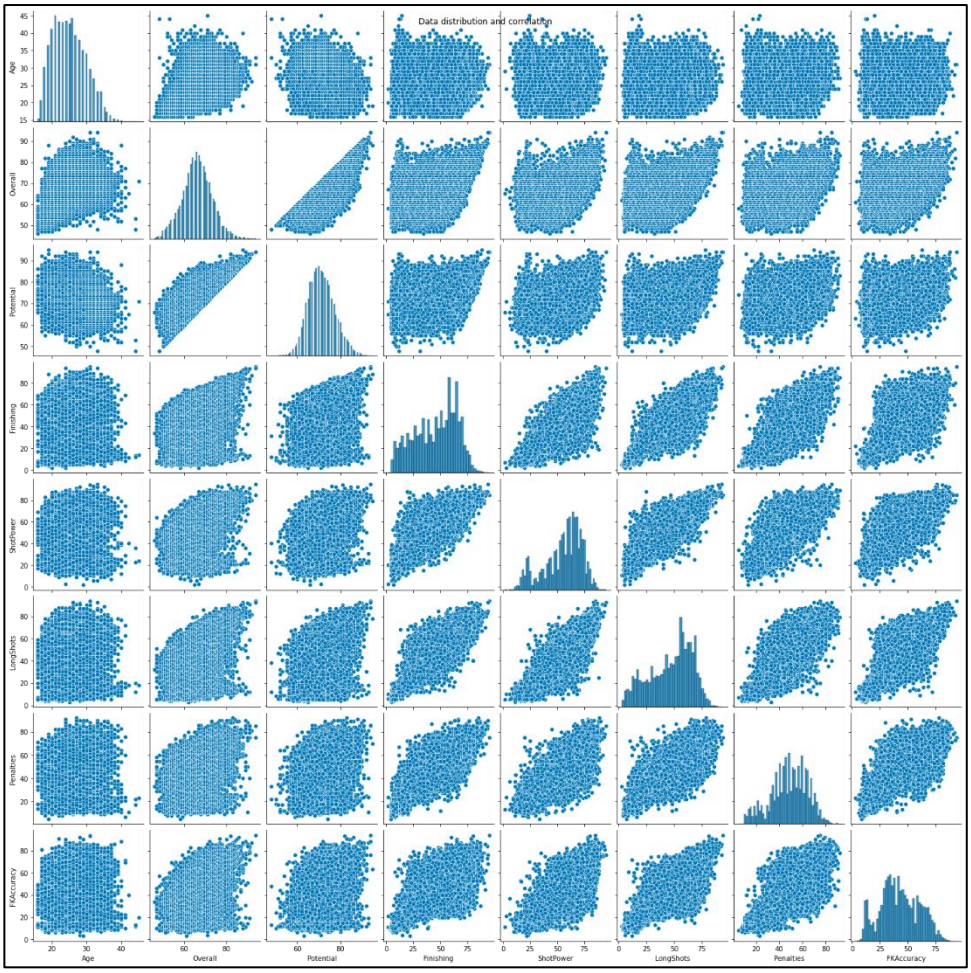


plot

We are looking at the pie chart that indicates the international reputation of players in Spain. Spain did not have a top rating for players, but we can apply the same results to other countries.

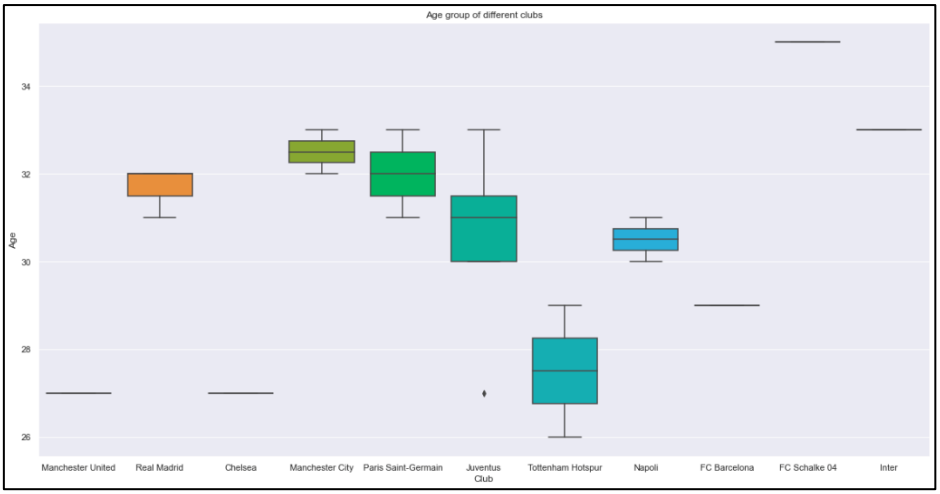
l. Pair plot:

With data ['Age', 'Overall', 'Potential', 'Finishing', 'Shot Power', 'LongShots', 'Penalties', 'FKAccuracy'], we did not see there is many linear regressions trend over scatterplot shown at the plot.



Plot

m. The age group of different clubs:

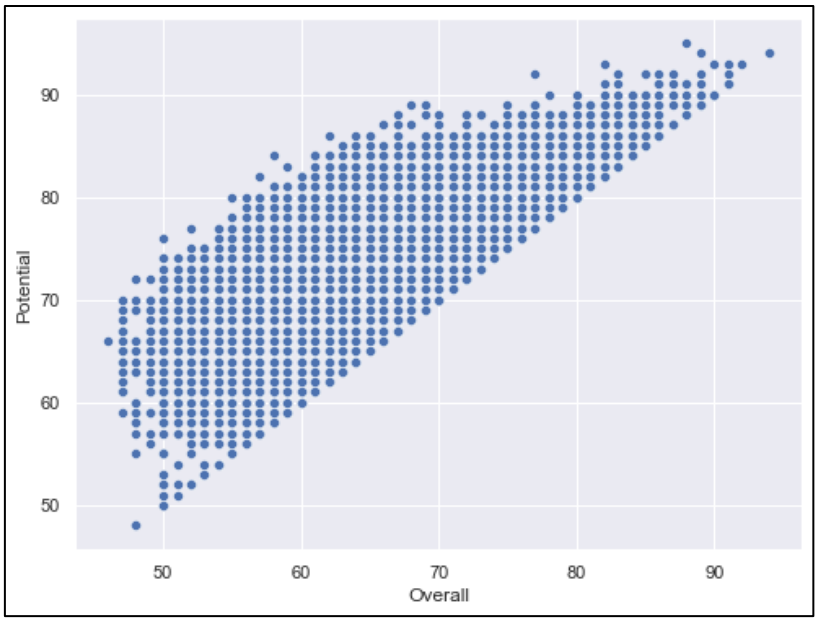


Plot

For the "Age group of different clubs", Manchester City has the oldest age group while Tottenham Hotspur has the youngest team.

Weak teams tend to be trained by youth teams, so the age structure is young, such as Tottenham. Ajax in the Netherlands is like this, especially Ajax trains young players and then is bought by giant clubs, which is like a player supermarket. Manchester City invests in "gold-dollar" football and has rich bosses, and then directly buys all kinds of big-name players. Some are older.

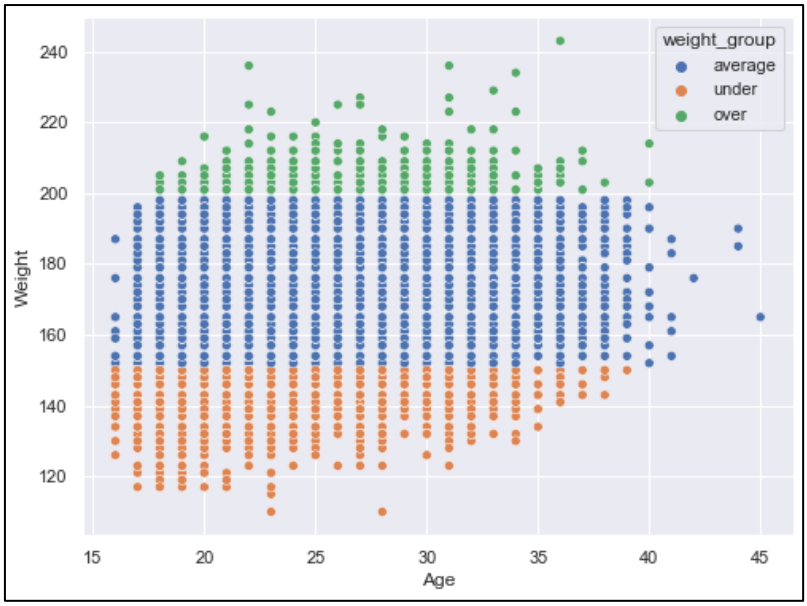
n. Scatterplot of player's potential:



Plot

Same as the histogram plot above, we can see that the scatter plot appears at the highest frequency at a rating of around 60-70. Two variables, overall rating, and potential are highly correlated. The extremely talented players get less as the overall rating is higher, which corresponds to reality.

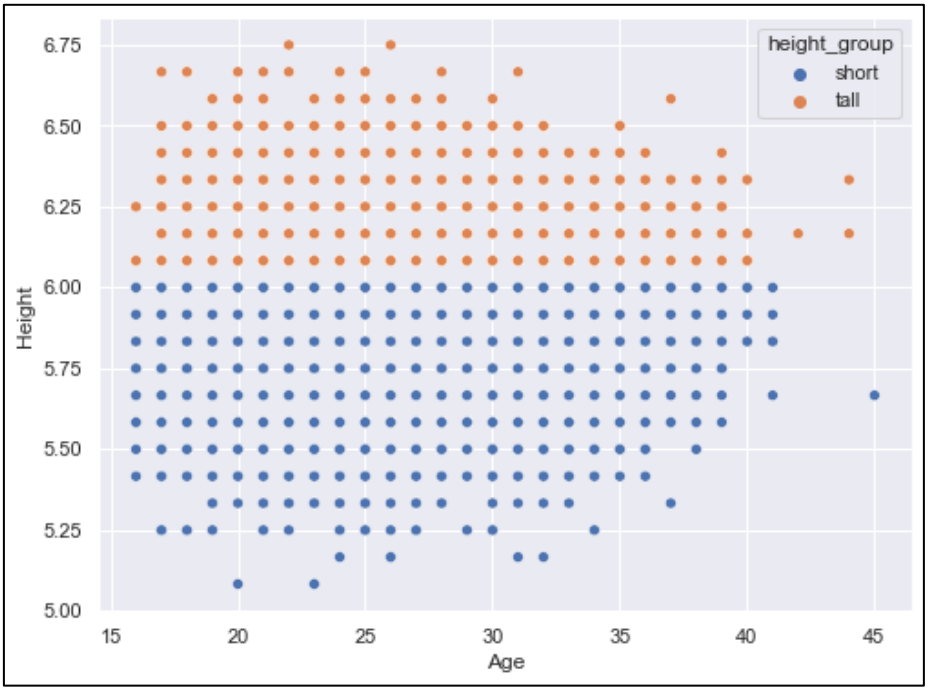
o. Scatterplot "age" vs. "weight":



Plot

For the scatterplot "Age" vs. "Weight", the age does not really correlate with weight. This graph will give us a sense what is the distribution of weight, which is around 160-180 lb.

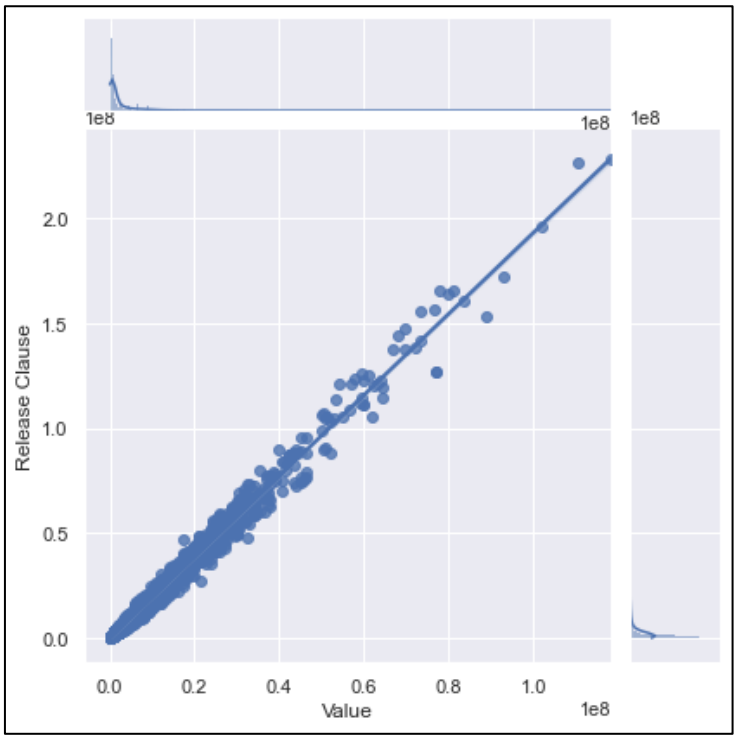
p. Scatterplot "Age" vs. "Height":



Plot

For the scatterplot "Age" vs. "Height", the distribution is evenly distributed. The height of 6 feet is divided by around half and half.

q. Relations between Value vs. Release Clause



Plot

There is a strong correlation between a player's release clause and value. The player's value is higher, and the buyout price is more expensive.

10. Classification Model:

We split 30% of the dataset to test data and 70% for the training dataset. In the data preparation part, we have divided our dataset into two parts, 1 is for high income and 0 is for low-income class:

```
data["Release_Clause_Class"] = data["Release_Clause_Class"].map({"low_income_class":0,
                                                                 "high_income_class":1})
```

Our target column will be 'Release_Clause_Class' and we drop that in our X columns. Setting 'Release_Clause_Class' as Y. The different classification models yield the different results as follows:

1. Logistic Regression

```
array([[2148,   59],
       [ 162, 2624]])
```

2. K Neighbors Classifier

```
array([[2052,  155],
       [ 229, 2557]])
```

3. Decision Tree Classifier

```
array([[2207,    0],
       [  95, 2691]])
```

4. Random Forest Classifier

```
array([[2207,    0],
       [  95, 2691]])
```

Specifically, for the random forest classification, it yields to an extremely high result as the accuracy of the test dataset:

```
accuracy on train :  1.0
auc on train :  1.0
accuracy on test :  0.9809733627077909
auc on test :  0.9829504666188082
```

For the precision rate, we can see out of the entire dataset, 96% of the low income are correctly predicted and 100% of high income are correctly predicted.

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	2207
1.0	1.00	0.97	0.98	2786
accuracy			0.98	4993
macro avg	0.98	0.98	0.98	4993
weighted avg	0.98	0.98	0.98	4993

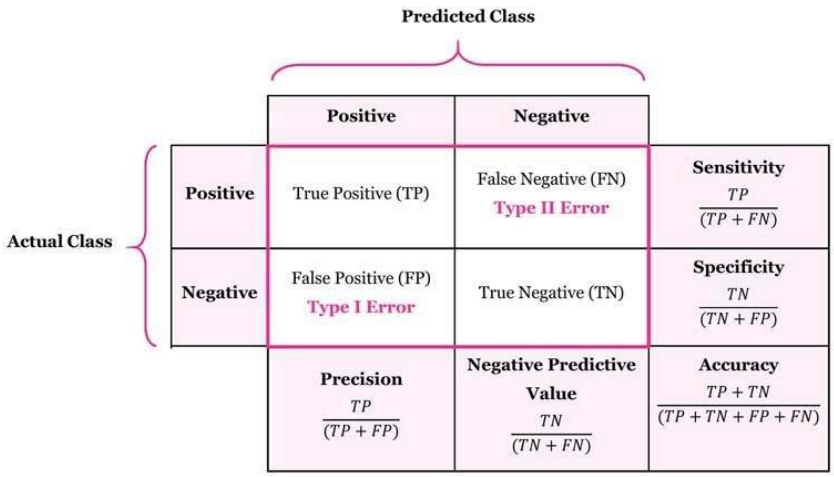
From the confusion matrix, the test set shows 2207+2691 number of incomes are corrected predicted:

```
confusion matrix train:
 [[5285    0]
 [   0 6364]]
confusion matrix test:
 [[2207    0]
 [  95 2691]]
```

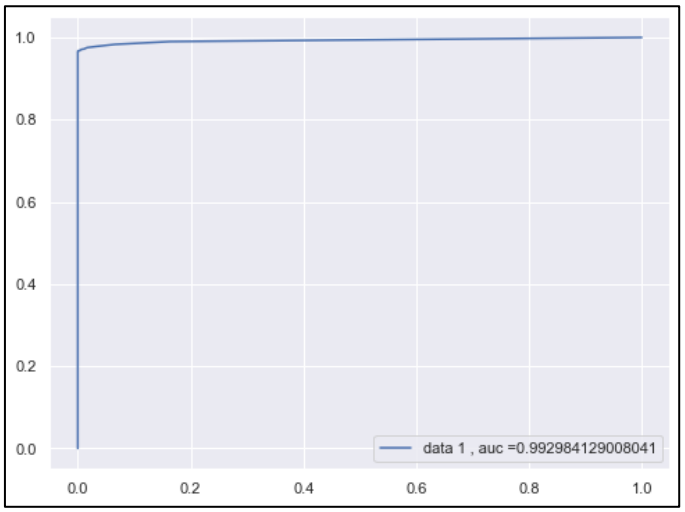
To be more specific, the TP is 2207, and TN is 2691. Type 1 error is 95 and type 2 error is 0:

```
total test set number: 4993
accuracy 0.9809733627077909
error 0.019026637292209093
sensitivity/Recall score: 0.9659009332376166
specificity: 1.0
precision: 1.0
```

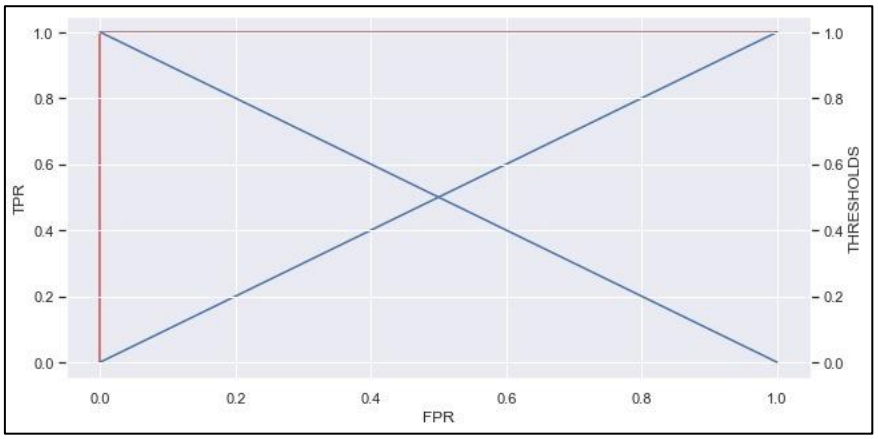
I will put this graph as a reference:



From the ROC curve, we can see the tp rate (also called sensitivity) is remarkably close to 1. From the previous result, we see the sensitivity/recall score is 0.9659009332376166. The AUC=99%m which means it is an ideal rate.



Plot



Plot

BOOSTING

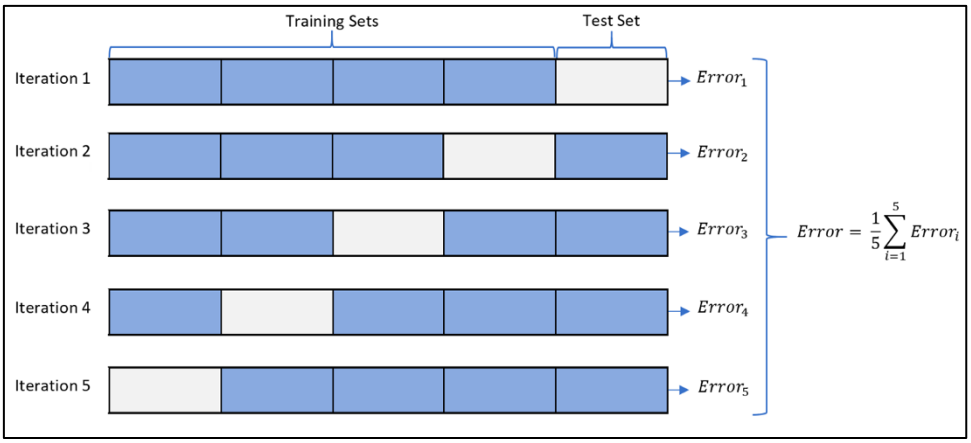
We use AdaBoostClassifier to boost our model. Boosting is useful especially when the dataset is small, and we want to avoid the problem of overfitting.

In this case, the boosting does not seem to affect the model very much:

```
accuracy on train :  1.0
auc on train :  1.0
confusion matrix :
 [[5285    0]
 [   0 6364]]
accuracy on test :  0.9809733627077909
auc on test :  0.9829504666188082
confusion matrix :
 [[2207    0]
 [  95 2691]]
```

MODELS COMPARISON

Lastly, we want to compare model performance by cross-validation.



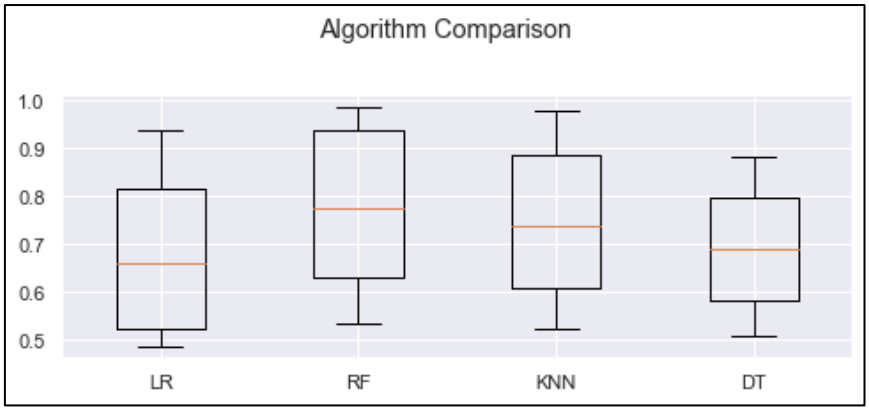
Plot

As shown in the results, the SD is small for all 4 models, so we can choose the one that has the highest mean:

```
'LR', LogisticRegression()))
'RF', RandomForestClassifier()))
'KNN', KNeighborsClassifier()))
'DT', DecisionTreeClassifier()))
```

```
LR: ('mean:', 0.6794605422730423) (('std:', 0.15930413665693363))
RF: ('mean:', 0.7759650756525757) (('std:', 0.1604294599422932))
KNN: ('mean:', 0.7466400294525293) (('std:', 0.15552933578483166))
DT: ('mean:', 0.6949086466273966) (('std:', 0.12348227779545826))
```

In this case, we will get the conclusion that the random forest gives the most ideal results. This also is more intuitively shown on a box plot:



Plot

As the graph shows that RF has the best performance, the decision tree gives the most stable distribution. In this case, the audience may decide which model best fits their needs.

11. Linear regression model:

In this part, we use the linear regression method to verify our data. The model results verified by the data are obtained.

First, we made the assumptions of the model.

- The Null Hypothesis: all columns have no relation to the target column release clause.
- Alternative Hypothesis: all columns have a relation with the target column release clause.

	df	sum_sq	mean_sq	F	PR(>F)
Preferred_Foot	1.0	9.211936e+13	9.211936e+13	12.270606	4.615968e-04
Work_Rate	8.0	6.436400e+16	8.045500e+15	1071.687445	0.000000e+00
Body_Type	9.0	1.434740e+17	1.594156e+16	2123.468358	0.000000e+00
Real_Face	1.0	3.323812e+17	3.323812e+17	44274.280951	0.000000e+00
LS	92.0	7.522404e+17	8.176527e+15	1089.140618	0.000000e+00
ST	92.0	1.411177e+17	1.533888e+15	204.318998	0.000000e+00
RS	92.0	5.935478e+16	6.451606e+14	85.937551	0.000000e+00
LW	104.0	1.153188e+17	1.108834e+15	147.700438	0.000000e+00
LF	101.0	2.325585e+16	2.302559e+14	30.670859	0.000000e+00
CF	101.0	1.603498e+16	1.587622e+14	21.147657	0.000000e+00
RF	101.0	1.254903e+16	1.242478e+14	16.550226	8.715449e-265
RW	104.0	1.951274e+16	1.876225e+14	24.991945	0.000000e+00
LAM	99.0	2.061221e+16	2.082041e+14	27.733482	0.000000e+00
CAM	99.0	5.757515e+15	5.815671e+13	7.746668	1.890903e-100
RAM	99.0	4.539872e+15	4.585729e+13	6.108344	1.217663e-71
LM	99.0	2.494914e+16	2.520115e+14	33.568768	0.000000e+00
LCM	91.0	1.247562e+16	1.370947e+14	18.261477	5.449000e-269
CM	91.0	3.627437e+15	3.986194e+13	5.309744	1.410508e-53
RCM	91.0	3.020967e+15	3.319744e+13	4.422010	5.523913e-40
RM	99.0	3.987367e+15	4.027644e+13	5.364956	6.666580e-59
LWB	94.0	2.530394e+16	2.691909e+14	35.857124	0.000000e+00
LDM	98.0	2.352521e+16	2.400532e+14	31.975881	0.000000e+00
CDM	98.0	2.025530e+15	2.066868e+13	2.753137	9.590235e-18
RDM	98.0	1.445686e+15	1.475190e+13	1.965002	4.633087e-08
RWB	94.0	1.684679e+15	1.792212e+13	2.387286	1.556629e-12
LB	97.0	6.937115e+15	7.151665e+13	9.526256	7.100463e-130
LCB	107.0	1.069029e+16	9.990925e+13	13.308245	3.320728e-216
CB	107.0	1.156577e+15	1.080913e+13	1.439812	2.064879e-03
RCB	107.0	1.231660e+15	1.151084e+13	1.533283	3.472934e-04
RB	97.0	1.584876e+15	1.633893e+13	2.176400	2.401565e-10
Residual	13880.0	1.042016e+17	7.507319e+12	NaN	NaN

We can see from the results of the data that some variables are significant, while some variables are not. To see which variables could be significant and which could not, we tested each variable.

```
data["Release_Clause"].mean()
```

```
4585060.986600974
```

```
x=data["Release_Clause"].sample(n=5000)
x.mean()
```

```
4796950.8
```

```
x=pd.DataFrame(x)
x.isna().sum()
```

```
Release_Clause    0
dtype: int64
```

```
x=x.dropna(how="all")
```

```
from scipy.stats import ttest_1samp,ttest_ind
from statsmodels.stats.power import ttest_power
print('Mean is %2.1f Sd is %2.1f' % (x["Release_Clause"].mean(),np.std(x["Release_Clause"],ddof = 1)))
```

```
Mean is 4796950.8 Sd is 11747045.6
```

```
t_statistic, p_value = ttest_1samp(x["Release_Clause"],4585060.98)
print(t_statistic, p_value)
```

```
1.275458819285844 0.2022061580191237
```

```
alpha=0.05
pvalue=p_value
if pvalue<alpha:
    print("reject null and accept alternate")
else:
    print("fail to reject null")
```

```
fail to reject null
```

WHEN POPULATION STANDARD DEVIATION IS NOT GIVEN

We looked at the significance of the variables through the concept of confidence intervals. Next, we did a two-sample test.

- h0: there is no difference in the release clause of players who are short and tall group : (mu_tall=mu_short)
- h1: there is no difference in the release clause of players who are short and tall group :(mu_tall!=mu_short)

We can see from the result that this result is insignificant and cannot reject the null hypothesis.

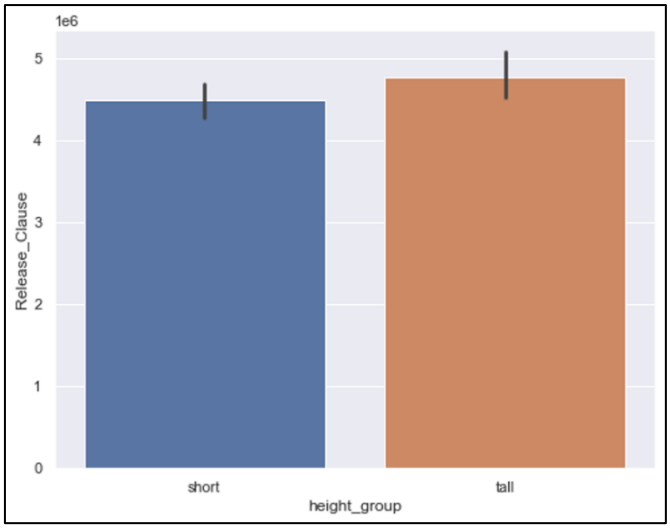
```
F_onewayResult(statistic=2.4869645622759378, pvalue=0.11481193892622046)
```

```
# using ols
formula="Release_Clause~height_group"
model=ols(formula,data).fit()
anova_table=anova_lm(model)
anova_table
```

	df	sum_sq	mean_sq	F	PR(>F)
height_group	1.0	3.074257e+14	3.074257e+14	2.486965	0.114812
Residual	16641.0	2.057075e+18	1.236148e+14	NaN	NaN

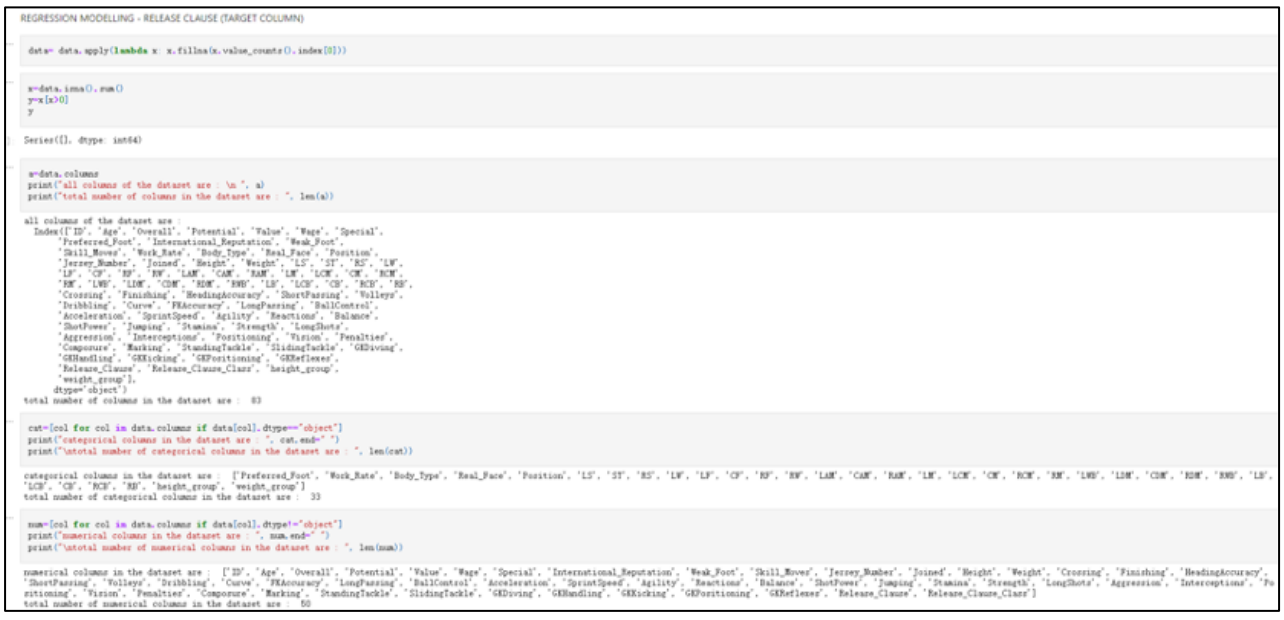
```
# Using two sample independent ttest
t,p=stats.ttest_ind(short,tall)
print("test_statistic : ",t)
print("p value : ",p)
```

```
test_statistic :  -1.577011275253272
p value :  0.11481193892688972
```

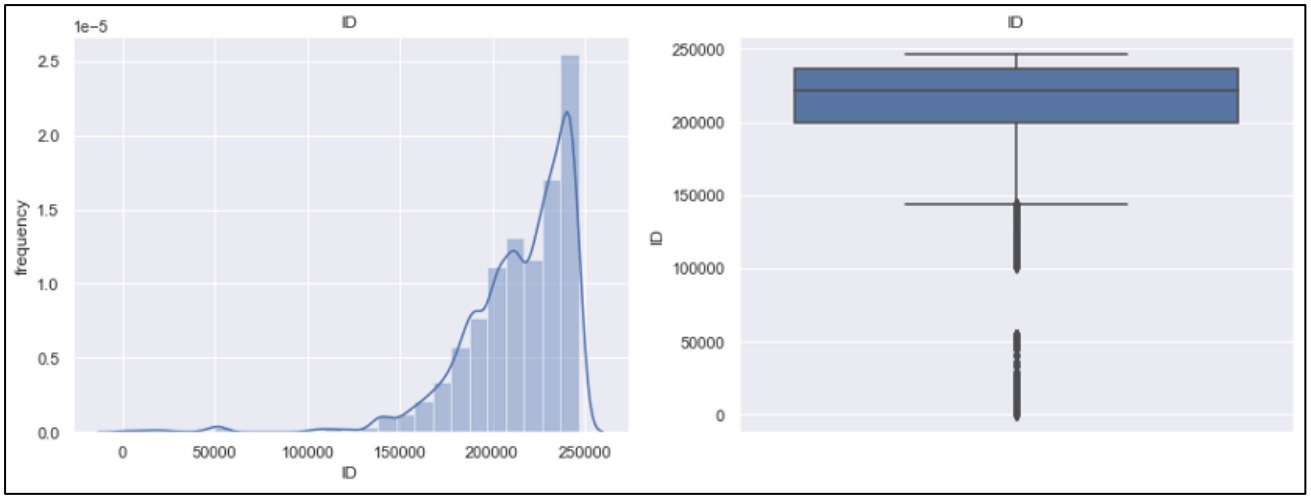


This means that there is no significant difference between the short group and the tall group. We also see through the CHI SQUARE test that, according to the results of confidence intervals, the Chi-square statistic is 2146.43783 and the P value is 0.000000, So we Reject the Null hypothesis and accept alternate.

Next, we set up the regression model. We set the hyperparameters and imported the data.



Looking at the skewness and distribution of the data variables, we can see that most of the data are stacked left and right into a normal distribution.



The dummy variable is treated in the same way as the dummy variable.

OLS Regression Results

Dep. Variable:	Release Clause	**R-squared:**	0.990
Model:	OLS	**Adj. R-squared:**	0.990
Method:	Least Squares	**F-statistic:**	1.679e+04
Date:	Mon, 15 Aug 2022	**Prob (F-statistic):**	0.00
Time:	14:12:02	**Log-Likelihood:**	-2.5536e+05
No. Observations:	16643	**AIC:**	5.109e+05
Df Residuals:	16545	**BIC:**	5.117e+05
Df Model:	97		
Covariance Type:	nonrobust		

We can see from the regression results that linear regression has a good fitting effect. After that, we do the feature selection.

```
#Backward Elimination
cols = list(X.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = X[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_with_p_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)
```

```
['Age', 'Overall', 'Potential', 'Value', 'Wage', 'International Reputation', 'Crossing', 'Finishing', 'ShortPassing', 'Volleys', 'LongPassing', 'Stamina',
'Body_Type_Messi', 'Real_Face_Yes', 'Position_CDM', 'Position_GK', 'Position_LF', 'Position_LS', 'Position_RAM', 'Position_RF', 'weight_group_over']
```

```
len((selected_features_BE))
```

```
27
```

We used filtered variables to build the model again.

```
#Adding constant column of ones, mandatory for sm.OLS model
X_constant = sm.add_constant(X)
#Fitting sm.OLS model
model = sm.OLS(y,X_constant).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	Release Clause	**R-squared:**	0.990
Model:	OLS	**Adj. R-squared:**	0.990
Method:	Least Squares	**F-statistic:**	6.030e+04
Date:	Mon, 15 Aug 2022	**Prob (F-statistic):**	0.00
Time:	14:12:06	**Log-Likelihood:**	-2.5540e+05
No. Observations:	16643	**AIC:**	5.108e+05
Df Residuals:	16615	**BIC:**	5.111e+05
Df Model:	27		
Covariance Type:	nonrobust		

As can be seen, the model fitting results are still exceptionally good. We used several types of models to fit the results of the regression. We used several types of models to fit the results of the regression.

The first model is a simple regression model

```
#MODEL 1 - SIMPLE LINEAR REGRESSION

X_train, X_test, y_train,y_test = train_test_split(X,y, test_size=0.3,random_state=42)
lr = LinearRegression()
lr.fit(X_train,y_train)
y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)
from sklearn.metrics import r2_score,mean_squared_error
print('R-squared on train data :',r2_score(y_train, y_pred_train))
print('R-squared on test data :',r2_score(y_test, y_pred_test))

print('\nRMSE on train data :', np.sqrt(mean_squared_error(y_train, y_pred_train)))
print('RMSE on test data :', np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
R-squared on train data : 0.9901742271819832
R-squared on test data : 0.9887766278808289

RMSE on train data : 1094986.677280849
RMSE on test data : 1195383.387125873
```

The second model is the random forest regression model

```
# MODEL 2 Random Forest
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
lr=RandomForestRegressor()
lr.fit(X_train,y_train)
y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)
from sklearn.metrics import r2_score,mean_squared_error
print('R-squared on train data :',r2_score(y_train, y_pred_train))
print('R-squared on test data :',r2_score(y_test, y_pred_test))

print('\nRMSE on train data :', np.sqrt(mean_squared_error(y_train, y_pred_train)))
print('RMSE on test data :', np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
R-squared on train data : 0.9979569361310865
R-squared on test data : 0.9875142446415004

RMSE on train data : 499305.61223493645
RMSE on test data : 1260819.5819979436
```

```
mse=np.mean((y_pred_test-y_test)**2)
rmse=np.sqrt(mse)
print("root_mean_squared_error :", rmse)
mae=(abs(y_pred_test-y_test)).mean()
print("mean_absolute_error :", mae)
```

```
root_mean_squared_error : 1260819.5819979436
mean_absolute_error : 434717.6587222111
```

```
ypred=pd.DataFrame(y_pred_test,columns=["predicted"])
ytest=pd.DataFrame(y_test)
ytest=ytest.reset_index(drop=True)
new=pd.concat([ytest,ypred],axis=1)
new
```

	Release Clause	predicted
0	8500000.0	9378000.0
1	11500000.0	9418000.0
2	2300000.0	2670000.0
3	1400000.0	1695000.0
4	998000.0	1014900.0

The third model is the decision tree regression model, and the fourth model is the Catboost regression model.s

```
# MODEL 3 Decision Tree
lr=DecisionTreeRegressor()
lr.fit(X_train,y_train)
y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)
from sklearn.metrics import r2_score,mean_squared_error
print('R-squared on train data :',r2_score(y_train, y_pred_train))
print('R-squared on test data :',r2_score(y_test, y_pred_test))

print('\nRMSE on train data :', np.sqrt(mean_squared_error(y_train, y_pred_train)))
print('RMSE on test data :', np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
R-squared on train data : 1.0
R-squared on test data : 0.9827141181642447

RMSE on train data : 0.0
RMSE on test data : 1483512.8231380344
```

```
# MODEL 4 Cat Boost Regressor
from catboost import CatBoostRegressor
lr=CatBoostRegressor()
lr.fit(X_train,y_train)
y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)
```

```
Learning rate set to 0.060347
0:      learn: 10488066.8373636 total: 56.3ms
1:      learn: 9978720.4772153  total: 58ms
2:      learn: 9475828.2125520  total: 59.6ms
3:      learn: 9006619.0458813  total: 61.3ms
4:      learn: 8581137.3867293  total: 62.9ms
5:      learn: 8152730.5626454  total: 64.9ms
```

```
from sklearn.metrics import r2_score,mean_squared_error
print('R-squared on train data :',r2_score(y_train, y_pred_train))
print('R-squared on test data :',r2_score(y_test, y_pred_test))

print('\nRMSE on train data :', np.sqrt(mean_squared_error(y_train, y_pred_train)))
print('RMSE on test data :', np.sqrt(mean_squared_error(y_test, y_pred_test)))
```

```
R-squared on train data : 0.9985854521040535
R-squared on test data : 0.9860854898443139

RMSE on train data : 415464.86699676275
RMSE on test data : 1331004.4139728015
```

```
mse=np.mean((y_pred_test-y_test)**2)
rmse=np.sqrt(mse)
print("root_mean_squared_error :", rmse)
mae=(abs(y_pred_test-y_test)).mean()
print("mean_absolute_error :", mae)
mape=(abs(y_test-y_pred_test)/y_pred_test).mean()
print("mean_absolute_percentage_error :",mape)
```

```
root_mean_squared_error : 1331004.413972802
mean_absolute_error : 415131.58686873293
mean_absolute_percentage_error : 0.13107357468387298
```

```
new["actual_predicted"]=new["predicted"].apply(lambda x: "%.7f" % x)
new["actual_predicted"]=new["actual_predicted"].astype(float)
new=new.drop(["predicted"],axis=1)
new["actual_predicted"]=new["actual_predicted"].apply(np.ceil)
new
```

	Release Clause	actual_predicted
0	8500000.0	9419283.0
1	11500000.0	10069783.0
2	2300000.0	2488346.0
3	1400000.0	1751202.0
4	998000.0	969406.0

Finally, we performed regularization and regression using RIDGE and LASSO.

```
#RIDGE
from sklearn.linear_model import Ridge
r=Ridge(alpha=0.05,normalize=True)
r.fit(multi_train_x,multi_train_y)
train_mae,train_mse,train_r2=predict_metrics(r,multi_train_x,multi_train_y)
test_mae,test_mse,test_r2=predict_metrics(r,multi_test_x,multi_test_y)
print(train_mae,train_mse,train_r2)
print(test_mae,test_mse,test_r2)
#RMSE
print(np.sqrt(train_mse))
print(np.sqrt(test_mse))
```

```
697279.9742344046 2113903278659.7104 0.9828774615285003
689906.2573848163 1935353730947.1836 0.9843915969046276
1453926.8477676965
1391169.9144774457
```

```
#LASSO
from sklearn.linear_model import Lasso
lm=Lasso(alpha=0.05,normalize=True)
lm.fit(multi_train_x,multi_train_y)
train_mae,train_mse,train_r2=predict_metrics(lm,multi_train_x,multi_train_y)
test_mae,test_mse,test_r2=predict_metrics(lm,multi_test_x,multi_test_y)
print(train_mae,train_mse,train_r2)
print(test_mae,test_mse,test_r2)
#RMSE
print(np.sqrt(train_mse))
print(np.sqrt(test_mse))
```

```
496892.8851416481 1242580887898.9478 0.9899351407078143
508154.7968439397 1280368656243.724 0.989673975471372
1114711.1230713308
1131533.7627502433
```

```
#ELASTIC NET
from sklearn.linear_model import ElasticNet
e=ElasticNet(alpha=0.05)
e.fit(multi_train_x,multi_train_y)
train_mae,train_mse,train_r2=predict_metrics(e,multi_train_x,multi_train_y)
test_mae,test_mse,test_r2=predict_metrics(e,multi_test_x,multi_test_y)
print(train_mae,train_mse,train_r2)
print(test_mae,test_mse,test_r2)
#RMSE
print(np.sqrt(train_mse))
print(np.sqrt(test_mse))
```

```
503083.0341100689 1331990352995.942 0.9884408717618929
486078.9179932556 1247072815386.6626 0.9912904098604755
1154118.8643272156
1116724.1447137527
```

We used different models to predict the regression model, and we used different models to choose the best one. It can be seen from the results obtained so far that Random Forest is the model with the best prediction effect. (R-squared on test data: 0.9875142446415004).

At the same time, after regularization, we can see that the accuracy of prediction has been improved, which indicates that our prediction ability has been improved after dimensionalities, and regularization is helpful for our prediction.

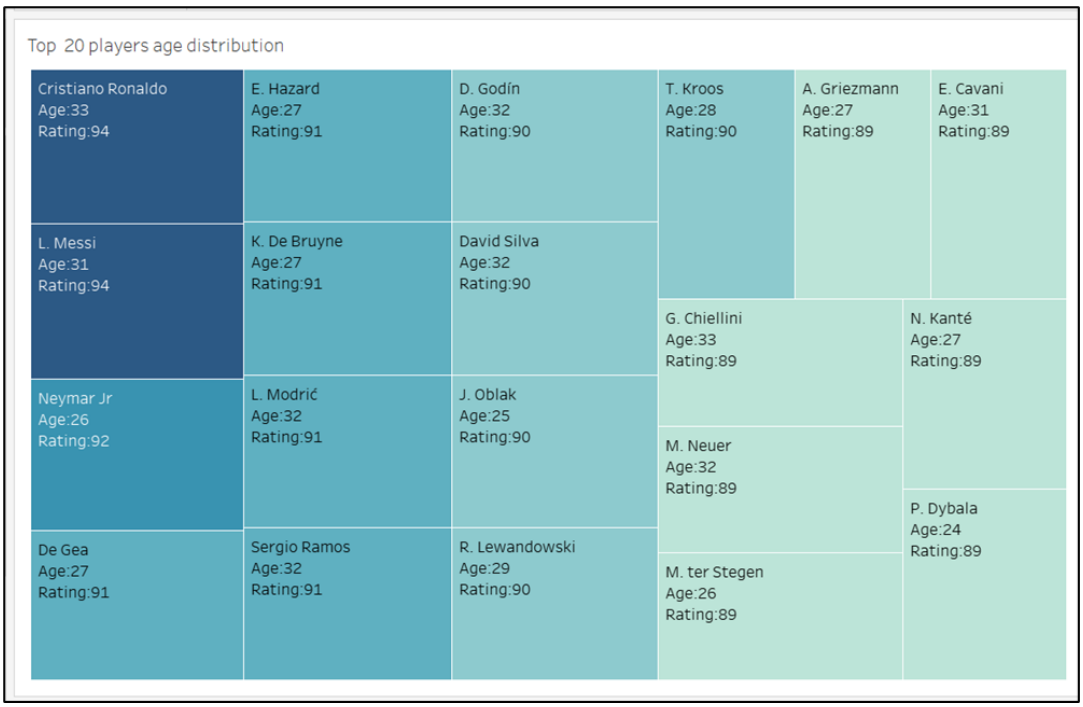
12. Tableau Dashboard for top 20 players based on Overall rating:

Using the tableau tool, we made the comparison of the top 20 players based on their Overall ratings given in the dataset. We compared using the following parameters :

- Age
- The position where they play
- Wage
- Their release clause
- Their jumping capability

a. **Age distribution :**

Player age is one of the leading factors in their performance and overall ratings, we plotted a graph for age distribution heat map as follows :



Age heatmap

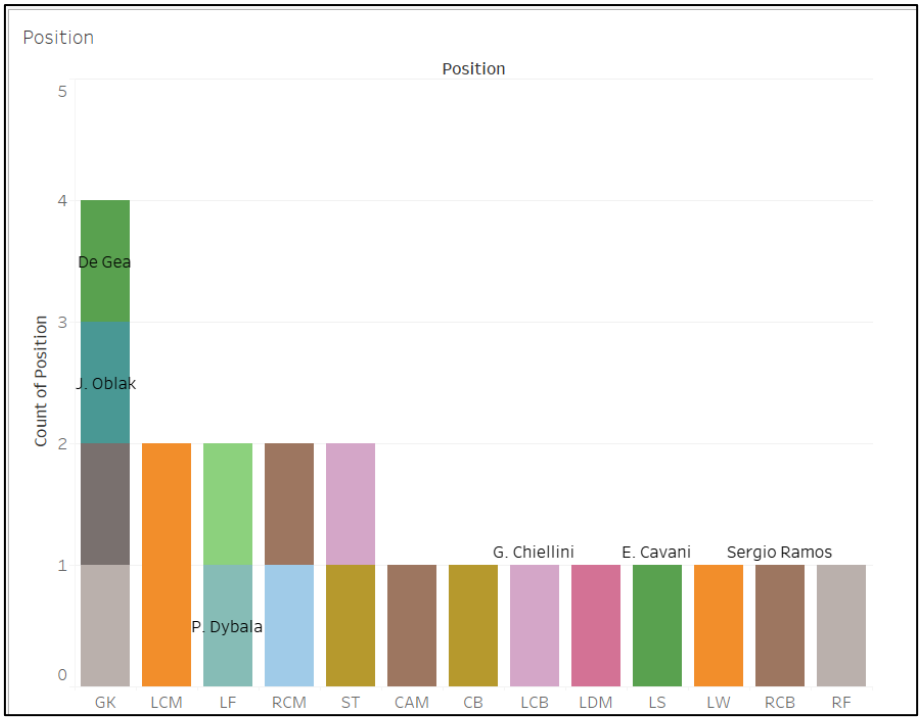
Interpretation :

Ronaldo and Messi got the highest rating and there both are in the almost same age group of 31-33

We can see that most of the players are under 33, which indicates that players are in their prime at a younger age

b. **Positions:**

Next, we saw the positions where these players play on the field :



Position plot

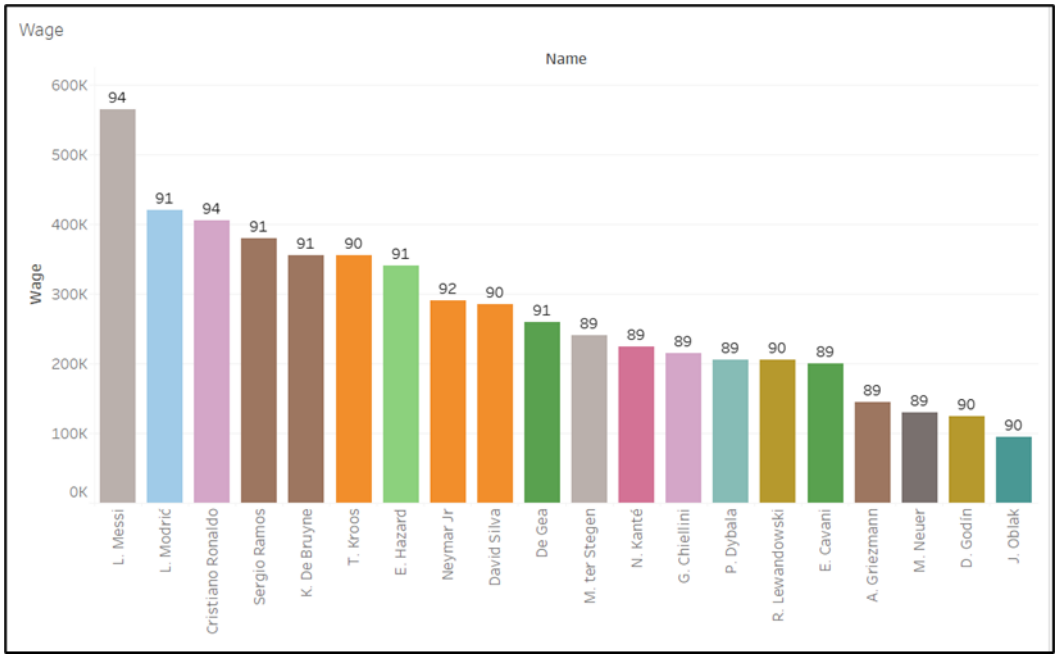
Interpretation :

- Four out of the top 20 players are Goalkeepers
- De Gea
- J. Oblak
- M.Neuer
- M. ter Stegen
- Ronaldo played at ST while Messi is at right forward
- 2 players are playing left forward, others play at the below position :
    - LCM : David Silva and Kroos
    - RCM : Modric and De Bruyne
    - CAM: Griezmann
    - CB: Godin
    - LCB: Chiellini
    - LDM: Kante

- LS: Cavani
- LW: Neymar Jr
- RCB: Ramos
- RF: Messi

c. **Wage:**

Next, we compared the wage of these players, along with their ratings :



Wage plot

Interpretation :

- Messi gets the highest wage of 565,000
- Oblak gets the lowest among these 20, which is 94,000
- We can see that mostly based on the overall rating, the wage of any player is also varying

d. **Release clause:**

Next, we compared the release clause value of these players,

Release Clause

Name	Release Clause
Neymar Jr	228,100,000
L. Messi	226,500,000
K. De Bruyne	196,400,000
E. Hazard	172,100,000
A. Griezmann	165,800,000
T. Kroos	156,800,000
P. Dybala	153,500,000
J. Oblak	144,500,000
De Gea	138,600,000
L. Modrić	137,400,000
Cristiano Ronaldo	127,100,000
R. Lewandowski	127,100,000
M. ter Stegen	123,300,000
N. Kanté	121,300,000
David Silva	111,000,000
E. Cavani	111,000,000
Sergio Ramos	104,600,000
D. Godín	90,200,000
M. Neuer	62,700,000
G. Chiellini	44,600,000

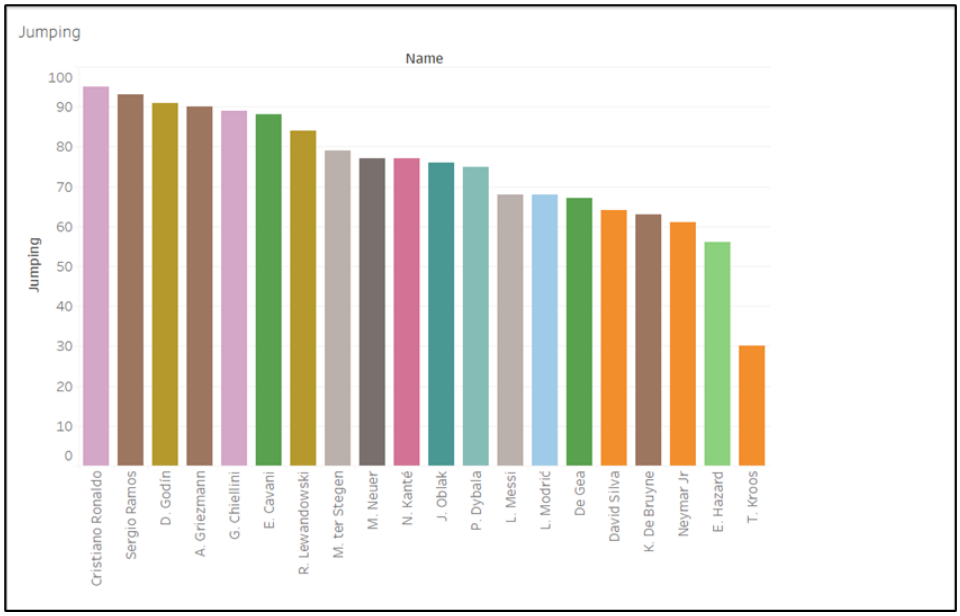
Wage plot

Interpretation :

- Neymar Jr is the highest earning among these top 20 with 228,100,000 USD (US Dollars)
- Messi is in second position with 226,500,000 USD
- Ronaldo, after ruling for a long time, is now on 11th position

e. **Jumping capability :**

Next, we compared the Jumping capability of these players,



Jumping plot

Interpretation :

- As we all know, Ronaldo is still holding the highest jumping record in FIFA sports, with his amazing fitness
- Next, we can see the goalkeepers are also having the good jumping skills which are required for saving the goal

f. Dashboard :

Using the above 5 graphs, I made an interactive dashboard where we can compare the performance of these 20 players. Below is a screenshot of the dashboard.

Top 20 Players comparison Dashboard

Summary

- From the analysis, we found the player having the highest sprint speed is Adama Traoré, Adebayo Akinfenwa Ranks 1st in strength and Lionel Messi is hold 1st position as the best finisher
- Lionel Messi is the highest-paid player
- Ramos and Chiellini record the same with 90 slide tackles
- The work rate for maximum players is medium, and the skill rating is 2. Ideally, both parameters should be maximum for satisfactory performance
- As we can see from the EDA analysis, most of the settings at FIFA 19 are highly corresponding to the real-world situation. For example, Manchester City has players among the top players in the world, more than 30 is just the year of playing. 32 to 33 are the peak period of mature players, which is shown in our EDA analysis
- For the Classification problem, we divide our data and group them into “low income” and “high income” and assign it to a binary question. our work shows the random forest yield the best result. However, the decision tree provides the evenest distribution. We may think twice about which model to use depending on the situation
- We use the linear regression method to verify our data. Then We looked at the significance of the variables through the concept of confidence intervals. Finally, we performed regularization and regression using RIDGE and LASSO

Bibliography

1. For NU (Northeastern University) logo:

   https://www.google.com/search?q=northeastern+university+logo+png

2. B. (2020, July 27). *FIFA World Cup: Complete Player's Analysis*. Kaggle. Retrieved August 16, 2022, from https://www.kaggle.com/code/blurredmachine/fifa-world-cup-complete-player-s-analysis

3. Performance measurement TP, TN, FP, FN are the parameters used in the ... (n.d.). Retrieved August 16, 2022, from https://www.researchgate.net/figure/Performance-measurement-TP-TN-FP-FN-are-the-parameters-used-in-the-evaluation-of_fig3_347447352

4. Patro, R. (2021, February 1). Cross-validation: K Fold vs Monte Carlo. Medium. Retrieved August 16, 2022, from https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b

5. *FIFA ratings explained: How is the overall rating created?* EarlyGame. (n.d.). Retrieved August 16, 2022, from https://earlygame.com/fifa/fifa-ratings-explained-overall-rating