Review 1

CSE3020 – Data Visualization

Slot: C1/C2

Faculty: Lydia Jane G

Visualization of Player Data in FIFA-19 game using Python and R

Group Members

1. Aditi Verma – 20BCE0813
2. Amul Thatai – 20BCE0837
3. Rishabh Manish Aggarwal – 20BCE802

Dataset Details

URL: https://www.kaggle.com/karangadiya/fifa19

Number of attributes: 88

Number of Rows: 18,207

Attributes Information

Our dataset consists of 88 attributes. The categorical attributes used to index the players are 'name' and 'player ID'. Player ID is unique for each player.

25 attributes are performance ratings of the player at a specific position. For example, the attribute 'GK' numerically denotes the ability of the player to perform as a Goalkeeper. Other such attributes are RM (right mid), LW(left wing), RF(right forward), and many more.

Data regarding age, nationality, monetary value, current club, lease period, joining date, etc. is also provided in the data set. Skill wise scores for all players are also provided in the dataset, for example, Crossing, Finishing, Dribbling etc.

All these attributes provide us with a detailed and extensive data set to analyze and finally find out what would be the 'perfect' FIFA-19 team.

Data Abstraction

Attribute Name	Attribute Type
ID	Categorical
Name	Categorical
Age	Numerical (Quantitative ordered)
Overall	Numerical (Quantitative ordered)
Potential	Numerical (Quantitative ordered)
International Reputation	Numerical (Quantitative ordered)
Height	Numerical (Quantitative ordered)
Weight	Numerical (Quantitative ordered)
Position Data (LS, GK, ST, RM, LW, LT)	Numerical (Quantitative ordered)
Skill Data (Dribbling, Passing, Volleying, Shooting, Agility)	Numerical (Quantitative ordered)
Club	Ordinally ordered
Nationality	Categorical
Image Data (Real Face, Photo, Flag, Club Logo)	Categorical Graphics

Task Abstraction

1. From which country are the majority of the players from in FIFA-19?
    a. Home country data of all players needed.
2. What is the distribution of age of players in FIFA-19?
    a. Age data of all players needed.
3. Which is the preferred foot of a greater number of players, left or right?
    a. Preferred foot data of all players needed.
4. How are the players distributed in terms of positions?
    a. Position data of all players needed.
5. Who are the top 10 players for each position?
    a. Positional score data of all players needed.
6. What is the financial distribution of resources among these players?
    a. Net value data of all players needed.
7. How is the overall rating distributed among all the players?
    a. Overall rating data of all players needed.
8. What is correlation between weight and height of players
    a. Weight and height data of all players needed
9. Obtain the attribute wise graphic of any player
    a. All details of a player needed
10. What is the best possible team that can be formed given some specific resources?
    a. All attributes need to be collectively analysed to determine this

11. Visualising skills of players
    a. Skill set data of all players

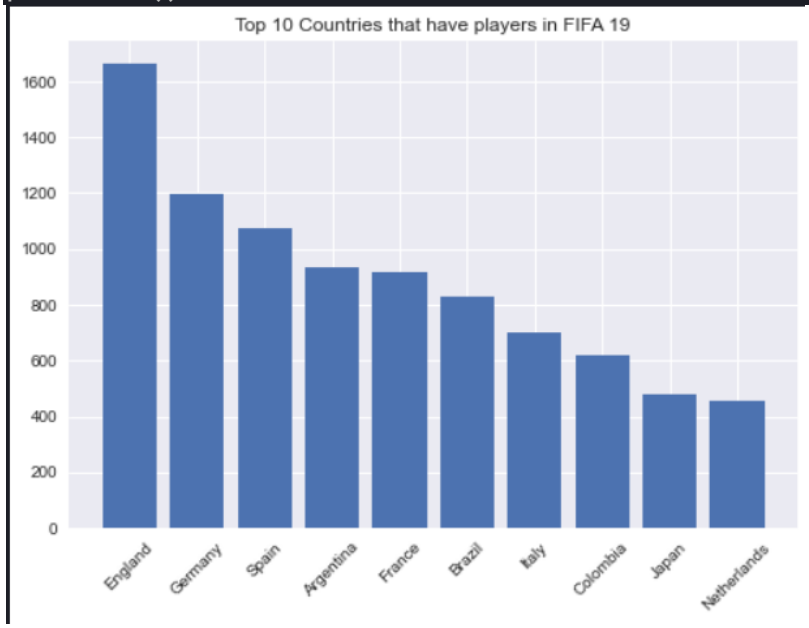
Encoding

```
# Import Modules
import warnings

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import scipy as sp
import seaborn as sns
from PIL import Image
from plotly.subplots import make_subplots
from scipy import stats
warnings.filterwarnings('ignore')

df = pd.read_csv("../Datasets/data.csv", index_col="Unnamed: 0")
df.head()
```

Q1

```
# Calculate top 10 countries sorted by most players in the game
# group data by Nationality and sort it by number of players to get
most countries having players.
national_players = df[['Nationality', "ID"]].groupby(by=['Nationalit
y'], as_index=False).count().sort_values(
    "ID", ascending=False)
national_players.rename(columns={'Nationality': "country", 'ID': 'pl
ayer_count'}, inplace=True)
national_players = national_players.reset_index()
national_players = national_players.drop(["index"], axis=1)
national_players.head(10)

# Slicing first 10 rows from country player_count dataset
player_count = national_players.iloc[0:10, 1]
nation = national_players.iloc[0:10, 0]

plt.style.use("seaborn")
# create bar chart
plt.bar(nation, player_count)
plt.xticks(rotation=45)
```

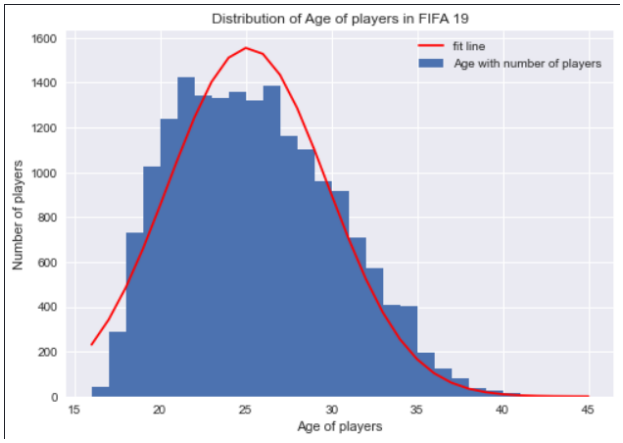
```
plt.title('Top 10 Countries that have players in FIFA 19')
plt.show()
```



Conclusion: It can hence be seen that England has the greatest number of players

Q2

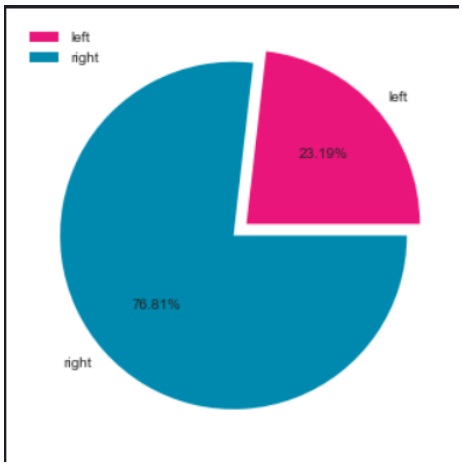
```
player_ages = df[['Age', "ID"]].groupby(by=['Age'], as_index=False).
count().sort_values("ID", ascending=False)
player_ages.rename(columns={'ID': 'count'}, inplace=True)
player_ages = player_ages.reset_index().drop(["index"], axis=1)
player_ages.head()
_, bins, _ = plt.hist(df.Age, bins=df.Age.max() - df.Age.min(), labe
l="Age with number of players")
mu, sigma = sp.stats.norm.fit(df.Age)
best_fit_line = sp.stats.norm.pdf(bins, mu, sigma)
plt.plot(bins, df.shape[0] * best_fit_line, label="fit line", color=
"red")

plt.title('Distribution of Age of players in FIFA 19')
plt.ylabel("Number of players")
plt.xlabel("Age of players")
plt.legend()
plt.show()
```



Conclusion: It can hence be concluded that the maximum number of players are middle aged, i.e., in the range of 25-30 years

Q3

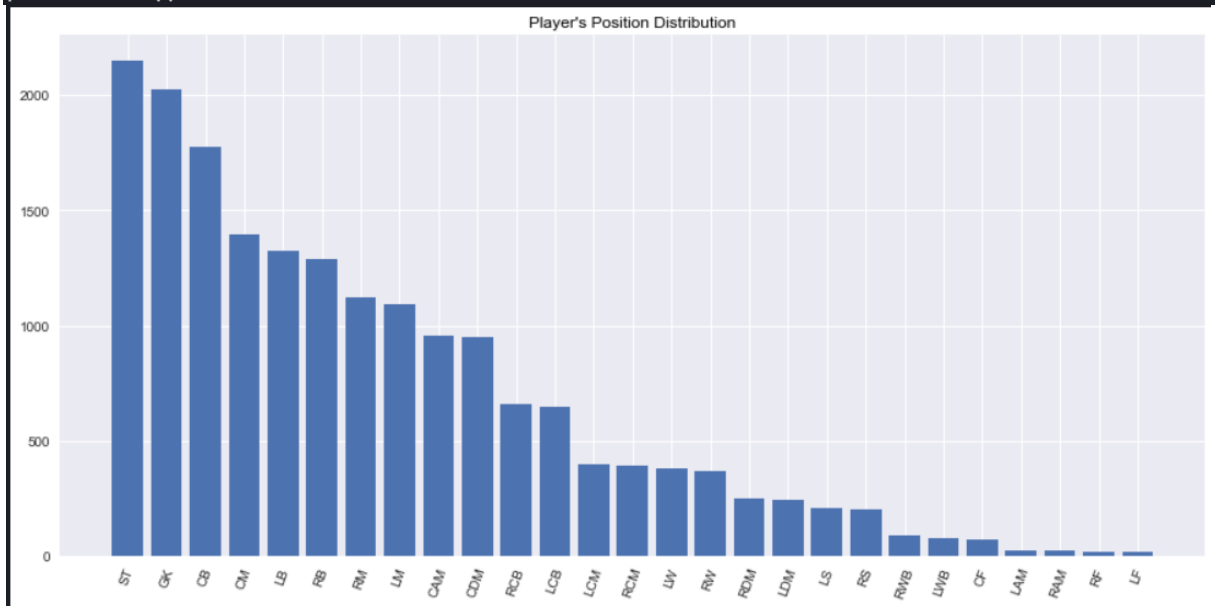
```
preferred_foot = df.groupby("Preferred Foot")["Preferred Foot"].coun
t()
print(preferred_foot)
plt.pie(preferred_foot, labels=["left", "right"], explode=[0.1, 0],
autopct='%1.2f%%', colors=["#ea157a", "#0089af"])
plt.legend()
plt.show()
```



Conclusion: It can be clearly seen that most of the players prefer their right foot (76.81%)

Q4

```
# plot bar chart to display the number of players for every position
plt.figure(figsize=(15, 7))
plt.bar(player_position["Position"], player_position["count"])
plt.xticks(rotation=70)
plt.title("Player's Position Distribution", color="black")
plt.show()
```

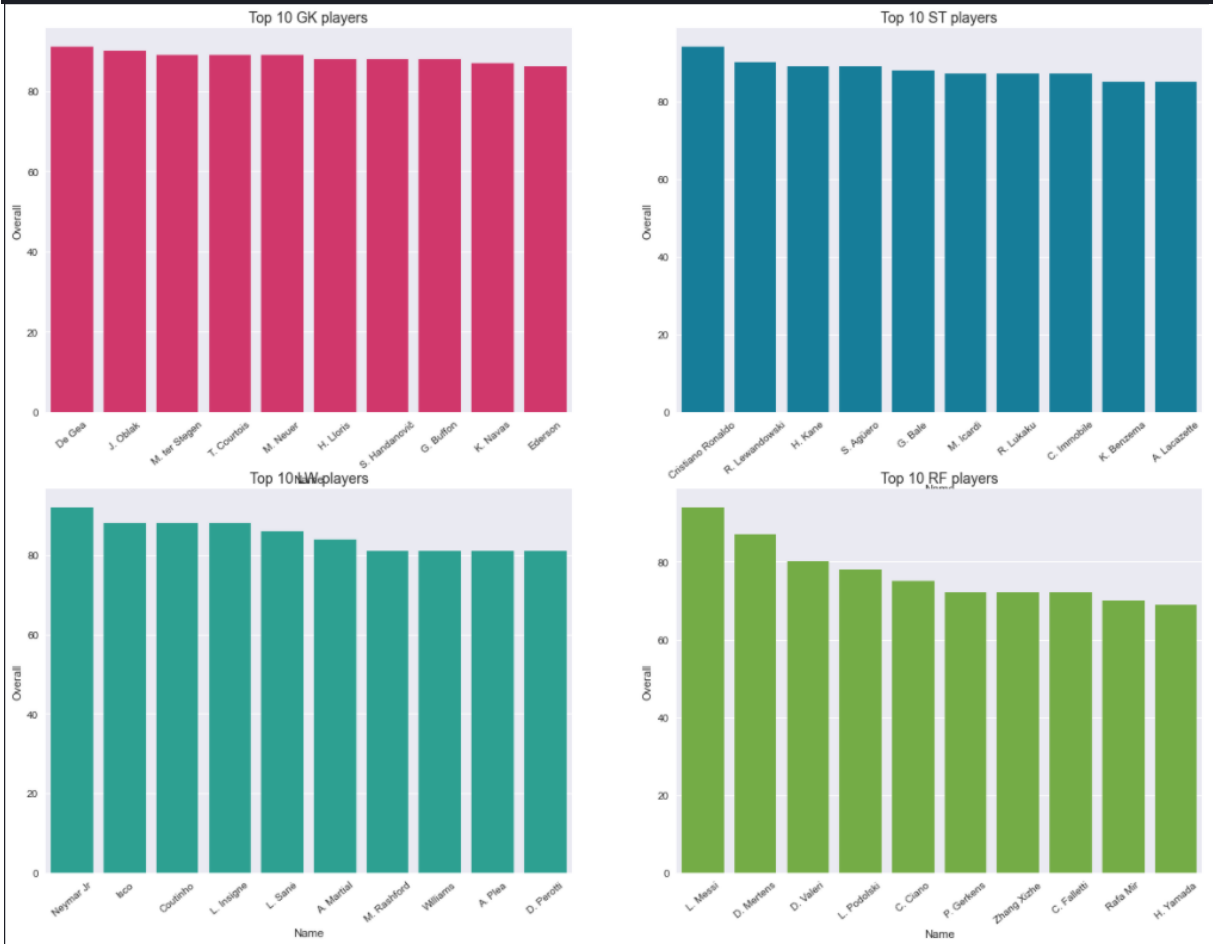


Conclusion: The plot shows the number of players at each position.

Q5

```
# Top 10 players for ST, GK, LW, RF Position

ST_position = df[df["Position"] == "ST"].sort_values("Overall", asce
nding=False)[["Name", "Overall"]]
ST_position = ST_position.iloc[:10, :]

GK_position = df[df["Position"] == "GK"].sort_values("Overall", asce
nding=False)[["Name", "Overall"]]
GK_position = GK_position.iloc[:10, :]

LW_position = df[df["Position"] == "LW"].sort_values("Overall", asce
nding=False)[["Name", "Overall"]]
LW_position = LW_position.iloc[:10, :]

RF_position = df[df["Position"] == "RF"].sort_values("Overall", asce
nding=False)[["Name", "Overall"]]
RF_position = RF_position.iloc[:10, :]
```

```
def draw(df, color, position, ax):
    plt.style.use('tableau-colorblind10')
    sns.barplot(df["Name"], df["Overall"], color=color, ax=ax).set_t
itle("Top 10 " + position + " players",

     fontsize=14)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=40)
# plot 4 figures that display Top 10 players in ST, GK, LW, RF posit
ions.
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=[20, 15])

draw(GK_position, "#e91e63", "GK", axes[0, 0])
draw(ST_position, "#0089af", "ST", axes[0, 1])
draw(LW_position, "#1ab39f", "LW", axes[1, 0])
draw(RF_position, "#72bd35", "RF", axes[1, 1])
plt.show()
```

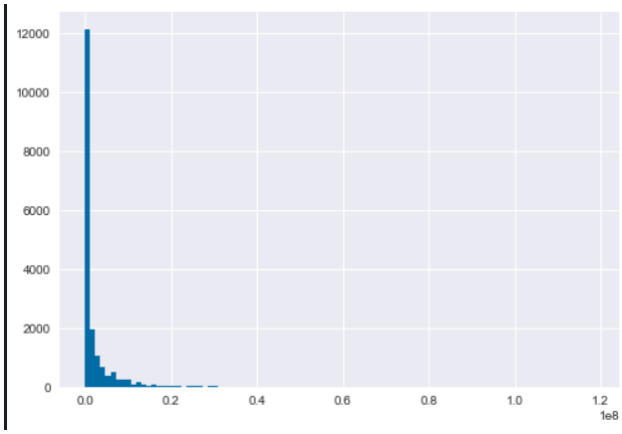


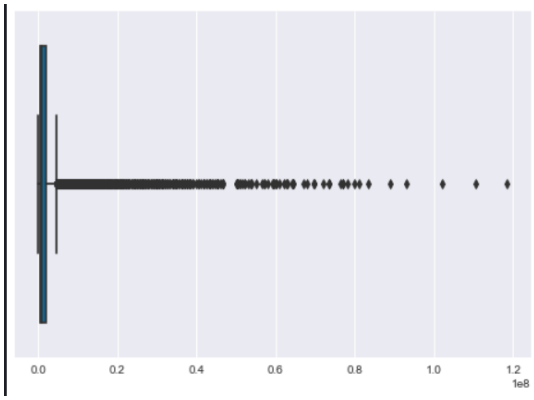
Conclusion: The plot shows the top players for each position, along with their overall scores.

Q6

```
# Distribution of all player's value and calculate The average value
 of players.
# function that converts value column of players to numeric.
def getValue(df):
    new = []
    for i in df:
        i = i.strip("€")
        if "K" in i:
            i = i.strip("K")
            new.append(float(i) * 1000)
        elif "M" in i:
            i = i.strip("M")
            new.append(float(i) * (10 ** 6))
        else:
            new.append(0.0)

    return new
# convert value columns to numeric and calculate the average value.
lis = getValue(df.Value.values)
print("The average value of players in the world = ", round(np.avera
ge(np.array(lis)) / 10 ** 6, 2), "M")
# plot histogram of values to show distribution of it.
plt.hist(lis, bins=100)
plt.show()
sns.boxplot(x=getValue(df.Value.values))
plt.show()
```

```
The average value of players in the world = 2.41 M
```

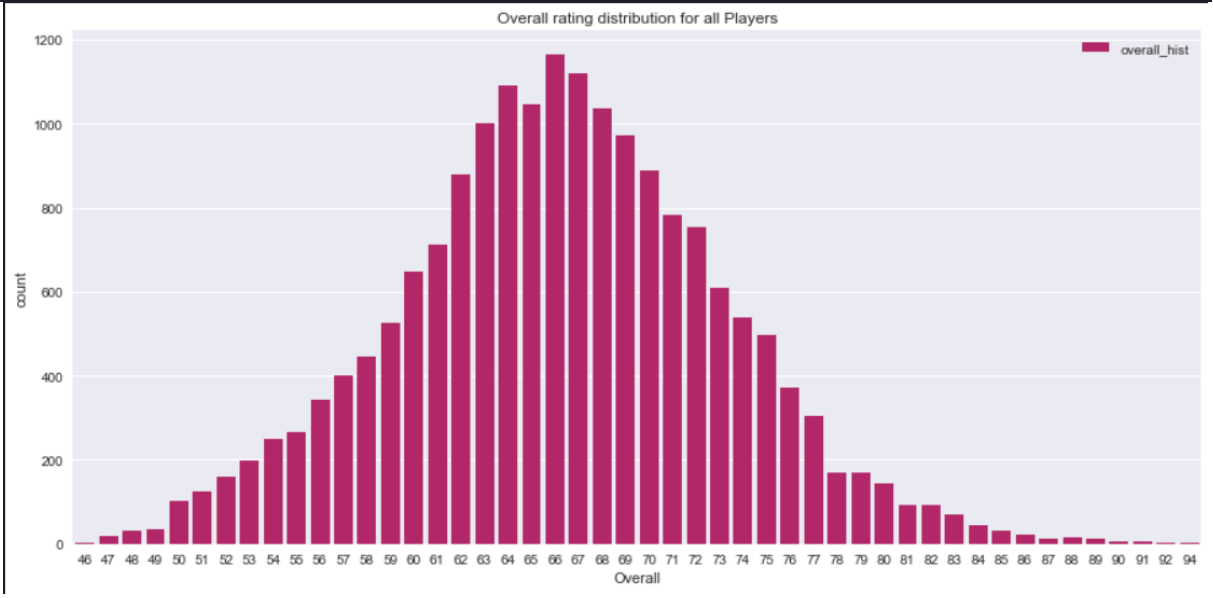




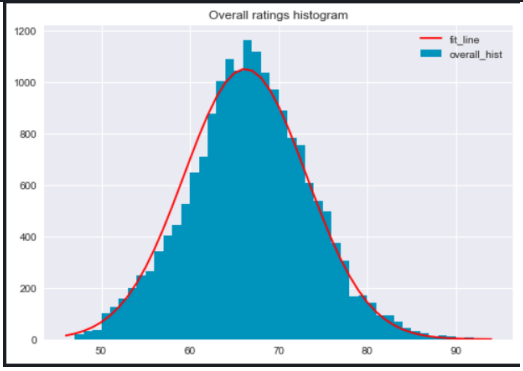
Conclusion: The plots show the distribution of monetary values of all players.

Q7

```
# Overall rating distribution and most fit line for it.
# plot the distribution of overall rating.
plt.figure(figsize=(15, 7))
sns.countplot(df.Overall, label="overall_hist", color="#c81067")
plt.title("Overall rating distribution for all Players")
plt.legend()
plt.show()
```



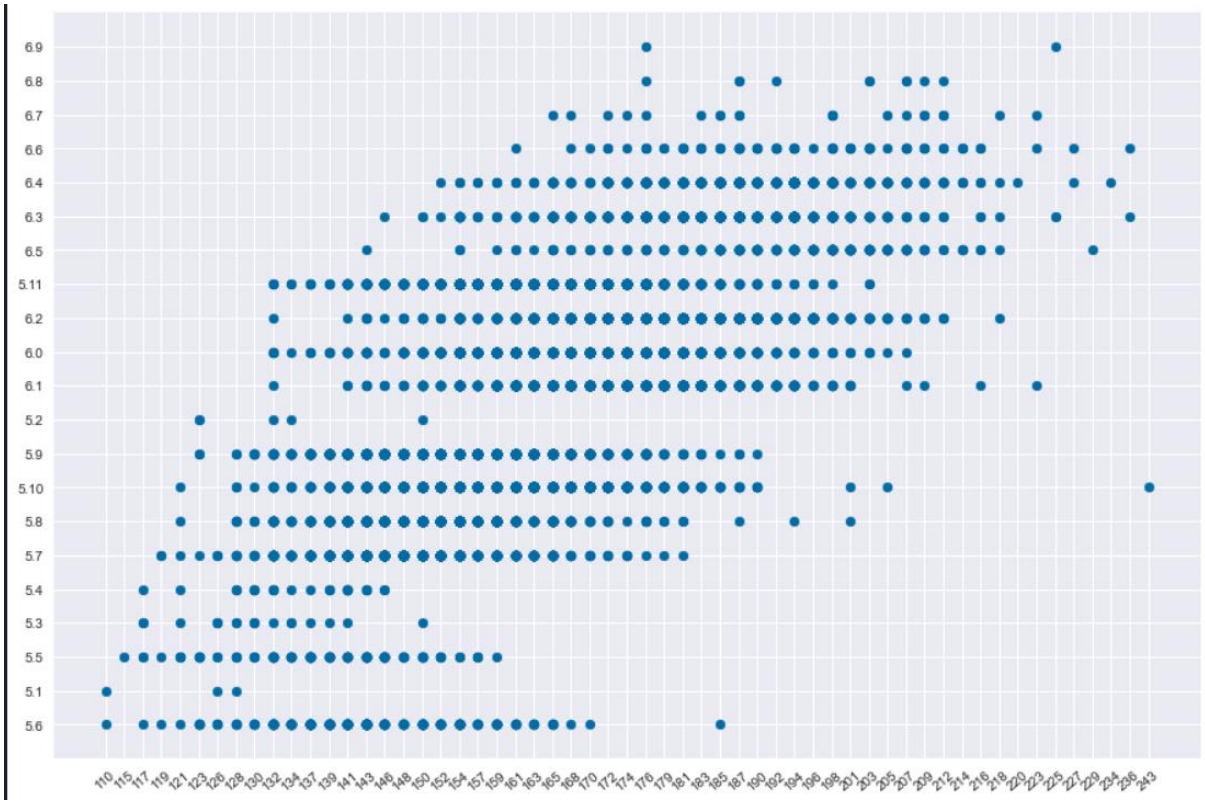

```
_, bins, _ = plt.hist(df.Overall, bins=(df.Overall.max() - df.Overal
l.min()), label="overall_hist", color="#0093bc")
mu, sigma = sp.stats.norm.fit(df.Overall)
best_fit_line = sp.stats.norm.pdf(bins, mu, sigma)
plt.plot(bins, df.shape[0] * best_fit_line, label="fit_line", color=
"red")
plt.title("Overall ratings histogram")
plt.legend()
plt.show()
```



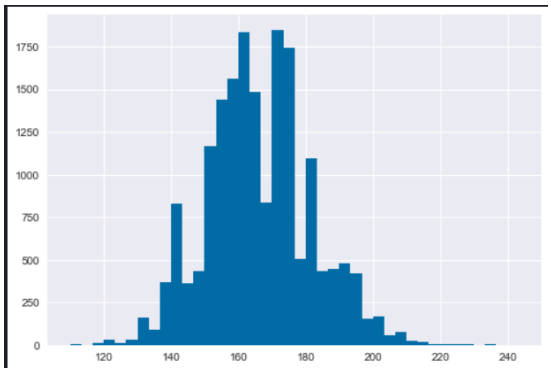
Conclusion: The 2 plots show how the overall ratings are distributed among the players.

Q8

```
# Calculate the relation between weight and height for all players(c
orr. and distribution)
# function that convert weight to numeric.
def get_weight(weight):
    new = []
    for i in weight:
        if "lbs" in str(i):
            new.append(i.strip("lbs"))
    return new
# function that convert height to numeric.
def get_height(height):
    new = []
    for i in height:
        if "'" in str(i):
            new.append(i.replace("'", "."))
    return new
# drop nan values form weight and height columns and convert it to n
umeric.
weight_height = df[["Weight", "Height"]].dropna(how="any")
weight_height.Weight = get_weight(list(weight_height.Weight.values))
weight_height.Height = get_height(list(weight_height.Height.values))
# plot scatter for  weight column.
weight_height.sort_values("Weight", ascending=True, inplace=True)
plt.figure(figsize=(15, 10))
plt.scatter(weight_height.Weight, weight_height.Height)
plt.xticks(rotation=40)
plt.show()

weight_height.Weight = weight_height.Weight.astype("float64")
weight_height.Height = weight_height.Height.astype("float64")
print("correlation between Weight and Height of players=", round(wei
ght_height.Weight.corr(weight_height.Height), 2))
```



```
# plot the distribution of weight.
plt.hist(weight_height.Weight, bins=40)
plt.show()
```



```
correlation between Weight and Height of players= 0.45
```

Conclusion: The correlation coefficient is obtained and the distribution is shown using plots.

Q9

```
attribute_dict = {"shooting": ["Positioning", "Finishing", "ShotPowe
r", "LongShots", "Volleys", "Penalties"],
```

```
                  "passing": ["Vision", "Crossing", "FKAccuracy", "S
hortPassing", "LongPassing", "Curve"],
                  "dribbling": ["Agility", "Balance", "Reactions", "
BallControl", "Dribbling", "Composure"],
                  "defending": ["Interceptions", "HeadingAccuracy",
"Marking", "StandingTackle", "SlidingTackle"],
                  "physical": ["Jumping", "Stamina", "Strength", "Ag
gression"]}
```

```
def calculate_attribute(dataframe, player_index):
    allcols = []

    for i in attribute_dict.values():
        allcols.extend(i)

    player_observation = dataframe.loc[player_index, allcols].astype
("int64")
    player_skills = []

    for i in attribute_dict.keys():
        lis = attribute_dict.get(i)
        player_skills.append(int(sum(player_observation[lis]) / len(
player_observation[lis])))

    return {i.upper() + ": " + str(j) + "%": j for i, j in zip(attri
bute_dict.keys(), player_skills)}
```

```
def get_attributes_values(attribute, observation):
    return observation.loc[attribute_dict.get(attribute)].astype("in
t64")
```

```
# function that plot player attribute: need index of player skills.
def plot_player_attribute(player_index, observation, skills):
    colors = ['#03a309', "#a3037e", "#fd3689", "#ded118", "#474bc9"]
    go.Figure()
    fig = make_subplots(rows=1, cols=5)
    # create skills bar chart
    for key, skill_name, color_i, column in zip(attribute_dict.keys(
), skills, colors, range(1, 6)):
        values = get_attributes_values(key, observation)
```

```
        fig.add_trace(go.Bar(x=values, y=attribute_dict.get(key), name=skill_name,
                             marker=go.bar.Marker(color=color_i, line=dict(color="#454545", width=1)), orientation="h",
                             width=0.5, text=values, textposition='auto'), row=1, col=column)

    # read image
    img = Image.open("../faces/" + str(player_index) + ".png")
    # Add image
    fig.add_layout_image(dict(source=img, xref="paper", yref="paper",
                              x=1, y=1.5, sizex=0.5, sizey=0.5, xanchor="right", yanchor="top"))
    # update layout properties
    fig.update_layout(autosize=False, height=300, width=2300, bargap=0.5, bargroupgap=0.3, barmode="overlay",
                      hovermode="x", margin=dict(r=0, l=0, b=0, t=100),
                      title=(
                          {'text': observation["Name"] + " ATTRIBUTE DETAILS", 'y': 0.9, 'x': 0.5, 'xanchor': 'right',
                           'yanchor': 'top'}))
    fig.update_xaxes(range=[0, 100])
    fig.show()
```

```
# Player Attributes select player index form dataset
# draw attribute details for MESSI.
player_index = 0
player_skills = calculate_attribute(df, player_index)
plot_player_attribute(player_index, df.iloc[player_index], list(player_skills.keys()))

# draw attribute details for RONALDO.
player_index = 1
player_skills = calculate_attribute(df, player_index)
plot_player_attribute(player_index, df.iloc[player_index], list(player_skills.keys()))

# draw attribute details for Neymar Jr
```

```
player_index = 2
player_skills = calculate_attribute(df, player_index)
plot_player_attribute(player_index, df.iloc[player_index], list(play
er_skills.keys()))

# draw attribute details for M.SALAH.
player_index = 26
player_skills = calculate_attribute(df, player_index)
plot_player_attribute(player_index, df.iloc[player_index], list(play
er_skills.keys()))
```

Conclusion: Player Attribute Details can hence be obtained and visualized in such a manner.

Q10

```
# function that get best squad in the world based on your Lineup.
def get_best_squad(lineup):
    best_squad = df[df.Position == "GK"].sort_values("Overall", asce
nding=False).iloc[0:1]
    for j, k in zip(position.keys(), range(3)):
        best = []
        for i in position.get(j):
            best.append(df[df.Position == i].sort_values(["Overall",
 "Potential"], ascending=[False, False]).iloc[0])
        best = pd.DataFrame(best).sort_values(["Overall", "Potential
"], ascending=[False, False])
```

```
        best = best.iloc[0:lineup[k]]
        best_squad = pd.concat([best_squad, best])
    return best_squad
```

```
# get best squad on the world based on lineup which you select.
best_squad = get_best_squad(lineup)
best_squad.reset_index(inplace=True)
player_index = list(best_squad.loc[:, ["index"]].values.reshape(11,
))
best_squad.drop("index", axis=1, inplace=True)
best_squad
```

```
# Plot the best squad on playground based on Lineup [3,4,3].
# location of player on chart.
location_3_4_3 = {0: [150, 80],
                  1: [150, 145],
                  2: [220, 145],
                  3: [80, 145],
                  4: [60, 300],
                  5: [150, 230],
                  6: [240, 300],
                  7: [150, 320],
                  8: [60, 400],
                  9: [150, 450],
                  10: [240, 400]
                  }
```

```
# Create figure
fig = go.Figure()

# Constants
img_width = 900  # 900
img_height = 1200  # 1200
scale_factor = 0.4

# add plot
# noinspection PyTypeChecker
fig.add_trace(
    go.Scatter(x=[0, img_width * scale_factor], y=[0, img_height * s
cale_factor], mode="markers", marker_opacity=0))
```

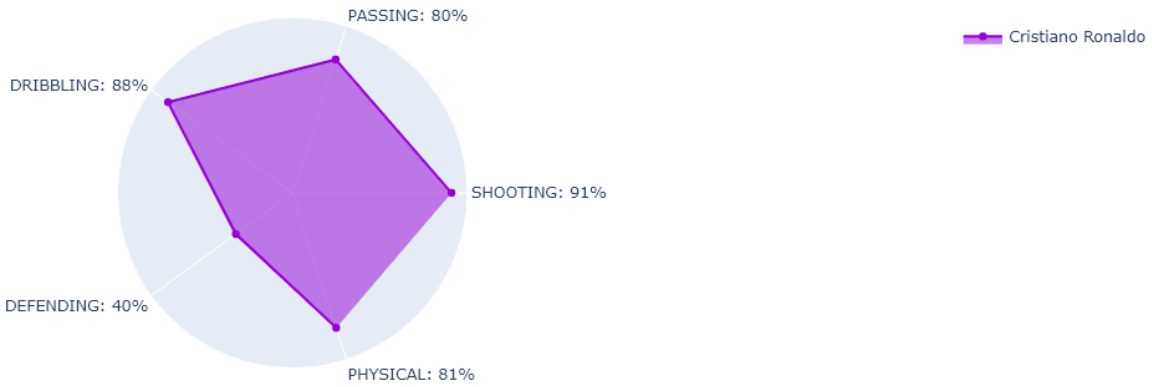
```
# Configure axes
fig.update_xaxes(visible=False, range=[0, img_width * scale_factor])

# disable y-axis visible
fig.update_yaxes(visible=False, range=[0, img_height * scale_factor]
, scaleanchor="x")

# add player image
for i in range(11):
    img = Image.open("../faces/" + str(i) + ".png")
    fig.add_layout_image(dict(x=location_3_4_3[i][0], y=location_3_4
_3[i][1], sizex=60, sizey=60, xref="x",
                              yref="y", opacity=1.0, layer="above",
source=img))
# Add background image
img = Image.open("../field.jpg")
fig.add_layout_image(dict(x=0, sizex=img_width * scale_factor, y=img
_height * scale_factor,
                          sizey=img_height * scale_factor, xref="x",
 yref="y", opacity=1.0, layer="below",
                          sizing="stretch", source=img))
# Configure other layout
fig.update_layout(width=img_width * scale_factor, height=img_height
* scale_factor,
                  margin={"l": 0, "r": 0, "t": 50, "b": 0},
                  title_font_size=15, title_font_family="Dosis",
                  title=({'text': "---
Best Squad in The World for Lineup[3,4,3]---",
                          'y': 0.95, 'x': 0.5, 'xanchor': 'center',
'yanchor': 'top', }))
fig.show()
```

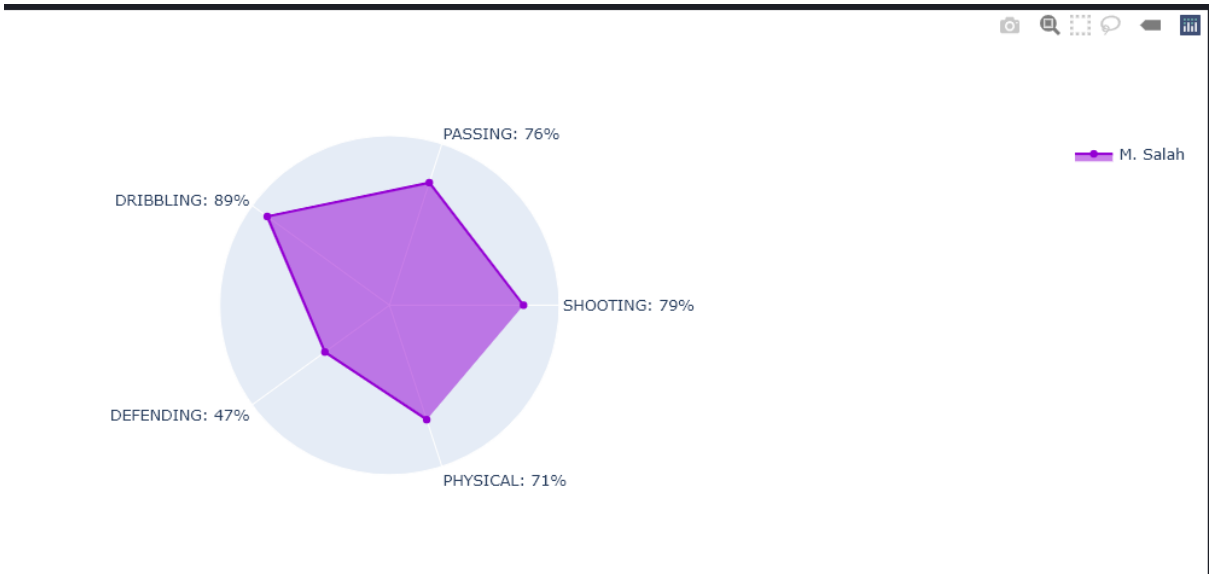
Conclusion: A graphical representation of the best squad possible can be obtained as show

Q11

```
# Radar Plot for player attribute's Details
def plot_player_radar(skills, player_name):
    fig = go.Figure()

    fig.add_trace(go.Scatterpolar(r=list(skills.values()), theta=list(skil
ls.keys()), fill='toself',
                                  name=player_name, line_color='darkviolet
', ))

    fig.update_layout(polar=dict(radialaxis=dict(visible=False, range=[0,
100])), showlegend=True)
    fig.show()
```

```
# draw attribute Details radar chart for RONALDO.


player_index = 1
player_skills = calculate_attribute(df, player_index)
plot_player_radar(player_skills, df.iloc[player_index]["Name"])


# draw attribute Details radar chart for M.SALAH.
player_index = 26
player_skills = calculate_attribute(df, player_index)
plot_player_radar(player_skills, df.iloc[player_index]["Name"])
```





The above RADAR graphs show the skill levels of the selected player