FEBRUARY 21, 2024

DATA ANALYTICS USING PYTHON

FOOTBALL STATISTICAL ANALYSIS

BASED ON DATA FROM THE 2021-22 CHAMPIONS LEAGUE SEASON

SAM - 1NH22AI147

TABLE OF CONTENTS

INTRODUCTION

WHAT IS STATISTICAL ANALYSIS IN FOOTBALL?

Statistical analysis in football, often referred to as football analytics or soccer analytics, involves the collection, interpretation, and application of data to gain insights into various aspects of the sport. This analysis typically covers a wide range of factors, including individual player performance, team tactics, match outcomes, player recruitment, injury prevention, and more. Statistical analysis in football relies heavily on data points such as goals scored, assists, passes completed, shots on target, possession percentages, and many other metrics.

Key aspects of statistical analysis in football include:

- **Performance Metrics**: Tracking various performance metrics for individual players and teams, such as goals scored, assists, successful tackles, interceptions, pass completion rates, and distances covered. These metrics help assess player and team effectiveness on the field.

- **Tactical Analysis**: Examining patterns of play, formations, and strategies employed by teams during matches. This analysis can reveal strengths and weaknesses in team tactics and inform adjustments to optimize performance.

- **Player Recruitment and Scouting**: Using statistical models to identify talented players for recruitment or scouting purposes. This may involve analyzing player statistics, performance trends, and potential contributions to a team's success.

- **Match Prediction**: Developing models to predict match outcomes based on historical data, team performance indicators, and other relevant factors. These predictions can be used for betting purposes or strategic planning by teams and coaches.

- **Injury Prevention and Player Fitness**: Analyzing player workload, injury history, and physical performance metrics to identify potential injury risks and optimize training regimes for injury prevention and player fitness.

- **Fan Engagement and Media Content**: Using statistical analysis to create engaging content for fans, such as infographics, data visualizations, and interactive tools that showcase interesting insights and trends in football.

Overall, statistical analysis plays a crucial role in modern football, helping teams, coaches, players, and fans make informed decisions and gain a deeper understanding of the sport.

WHAT IS THE NEED FOR STATISTICAL ANALYSIS IN FOOTBALL?

Statistical analysis in football serves several important purposes, addressing various needs within the sport:

- **Performance Evaluation**: Statistical analysis provides objective measures of player and team performance. By tracking key metrics such as goals scored, passes completed, tackles won, and distances covered, coaches and managers can assess the effectiveness of players and teams, identify areas for improvement, and make informed decisions about tactics and player selections.

- **Tactical Insights**: Analyzing match data allows coaches to gain insights into opponents' playing styles, strengths, and weaknesses. This information can inform strategic decisions regarding formations, game plans, and in-game adjustments to exploit opponents' vulnerabilities and maximize chances of success.

- **Player Recruitment and Scouting**: Statistical analysis plays a crucial role in player recruitment and scouting processes. By evaluating player performance metrics, potential contributions to team success, and suitability for specific roles or playing styles, clubs can identify and recruit talented players who align with their strategic objectives and budget constraints.

- **Strategic Planning and Decision-Making**: Statistical models and predictive analytics help teams forecast match outcomes, assess risks, and make data-driven decisions. This includes decisions related to player transfers, contract negotiations, youth development, and investment in facilities and infrastructure.

- **Injury Prevention and Player Fitness**: Monitoring players' physical performance metrics and workload helps teams identify injury risks, optimize training regimes, and maximize player fitness and availability over the course of a season. This can reduce the likelihood of injuries and improve team performance on the field.

- **Competitive Advantage**: In a highly competitive environment, clubs are constantly seeking ways to gain an edge over their rivals. Statistical analysis provides a means of extracting actionable insights from vast amounts of data, enabling clubs to make more informed decisions and potentially outperform competitors on the field.

Overall, statistical analysis is essential in football for enhancing performance, strategic planning, talent identification, fan engagement, injury prevention, and gaining a competitive advantage in an increasingly data-driven sport.

THE DATASETS USED

For this project I have used a dataset from kaggle.com which consists of all the important and basic stats of all the players who took part in the 2021-2022 champion league season, a season that was ultimately concluded on the 29th of may, 2022 with real madrid winning in the final game against opposition Liverpool courtesy of a Vinicius junior goal

The dataset: https://www.kaggle.com/datasets/azminetoushikwasi/ucl-202122-uefa-champions-league/code?datasetId=2226795

The dataset consists of 8 CSV files all containing a variety of player stats and different metrics which are all taken into account in this report

In addition to this, I have also added another dataset which contains all the details of the games played and their results in order to be able to understand the differences to which each team performed in the given champions league season as I analyse the individual team performances as well

This dataset can be found here:

https://fixturedownload.com/results/champions-league-2021

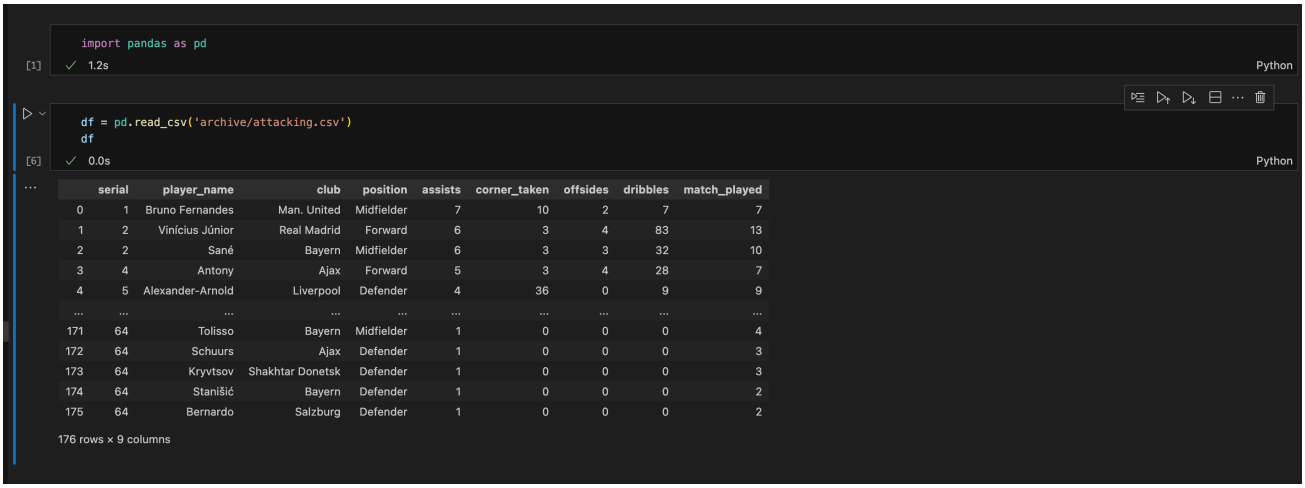
This site provides csv files which you can download for the results of every fixture in the given campaign

THE DATASETS

For this project, I have used a total of 9 csv files for comparing players of different positions on the basis of different metrics

The files are as follows:

1. ATTACKING.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

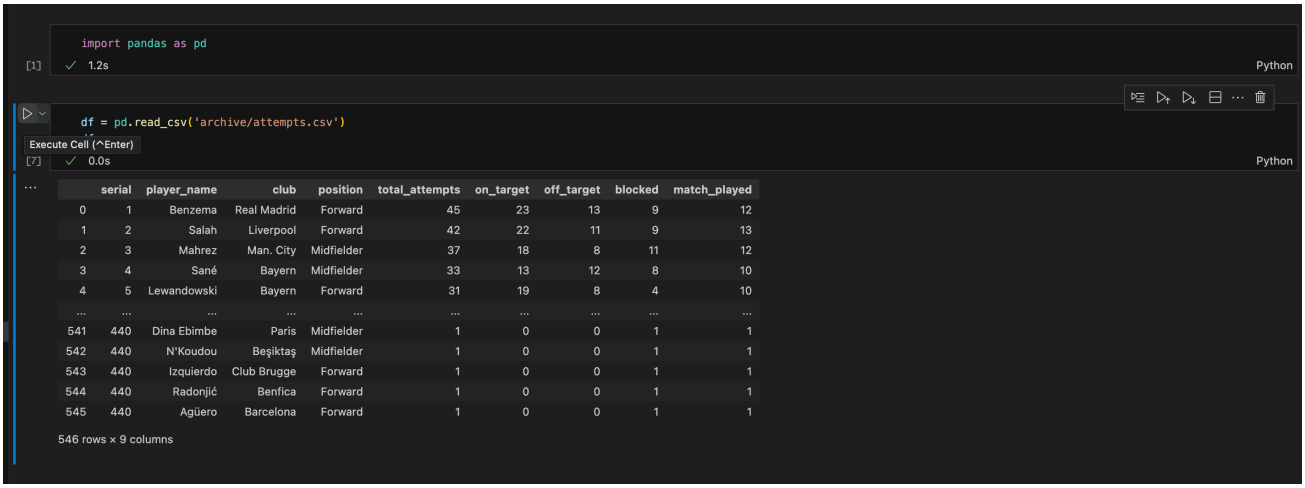
```
df = pd.read_csv('archive/attacking.csv')
df
```
[6] ✓ 0.0s Python

	serial	player_name	club	position	assists	corner_taken	offsides	dribbles	match_played
0	1	Bruno Fernandes	Man. United	Midfielder	7	10	2	7	7
1	2	Vinícius Júnior	Real Madrid	Forward	6	3	4	83	13
2	2	Sané	Bayern	Midfielder	6	3	3	32	10
3	4	Antony	Ajax	Forward	5	3	4	28	7
4	5	Alexander-Arnold	Liverpool	Defender	4	36	0	9	9
...	...	...	...	...	...	...	...	...	...
171	64	Tolisso	Bayern	Midfielder	1	0	0	0	4
172	64	Schuurs	Ajax	Defender	1	0	0	0	3
173	64	Kryvtsov	Shakhtar Donetsk	Defender	1	0	0	0	3
174	64	Stanišić	Bayern	Defender	1	0	0	0	2
175	64	Bernardo	Salzburg	Defender	1	0	0	0	2

176 rows × 9 columns

As we can see, this is the attacking.csv file and it contains 9 columns and 176 rows with stats like player name, club, position, assists, corners taken, offside, dribbles and matches played therefore mainly pertaining to the forward players and the opponents half of the field where they are involved.

2. ATTEMPTS.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

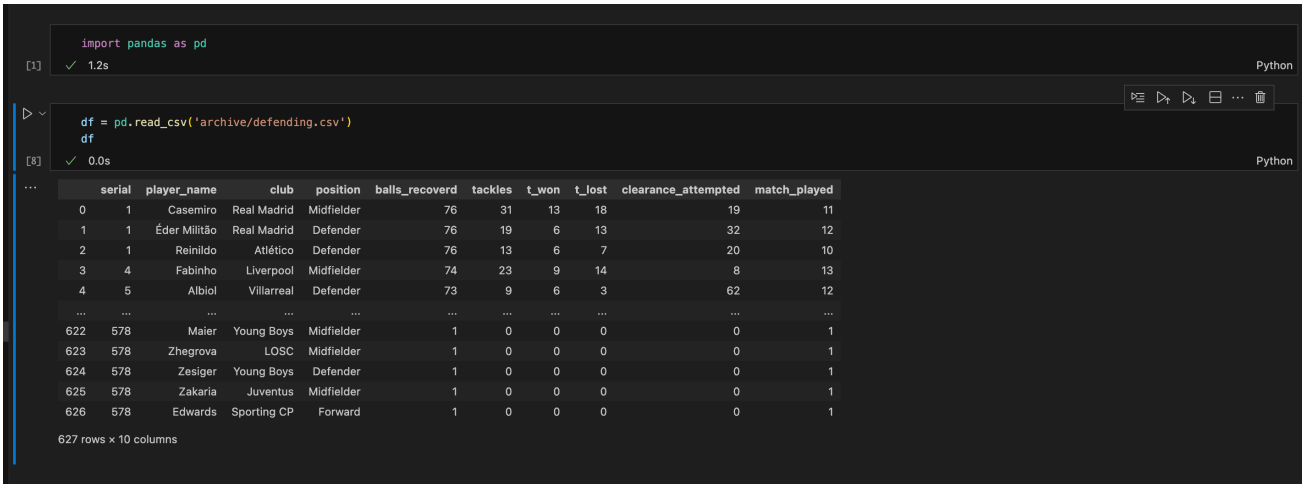
```
df = pd.read_csv('archive/attempts.csv')
```
Execute Cell (^Enter)
[7] ✓ 0.0s Python

	serial	player_name	club	position	total_attempts	on_target	off_target	blocked	match_played
0	1	Benzema	Real Madrid	Forward	45	23	13	9	12
1	2	Salah	Liverpool	Forward	42	22	11	9	13
2	3	Mahrez	Man. City	Midfielder	37	18	8	11	12
3	4	Sané	Bayern	Midfielder	33	13	12	8	10
4	5	Lewandowski	Bayern	Forward	31	19	8	4	10
...	...	...	...	...	...	...	...	...	...
541	440	Dina Ebimbe	Paris	Midfielder	1	0	0	1	1
542	440	N'Koudou	Beşiktaş	Midfielder	1	0	0	1	1
543	440	Izquierdo	Club Brugge	Forward	1	0	0	1	1
544	440	Radonjić	Benfica	Forward	1	0	0	1	1
545	440	Agüero	Barcelona	Forward	1	0	0	1	1

546 rows × 9 columns

The next csv file is called attempts.csv and as we can see, this file contains 9 columns and 546 rows with stats like player name, club, position, total attempts, attempts on target, attempts off target, attempts blocked and matches played therefore again mainly pertaining to the forward players and their shots

3. DEFENDING.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

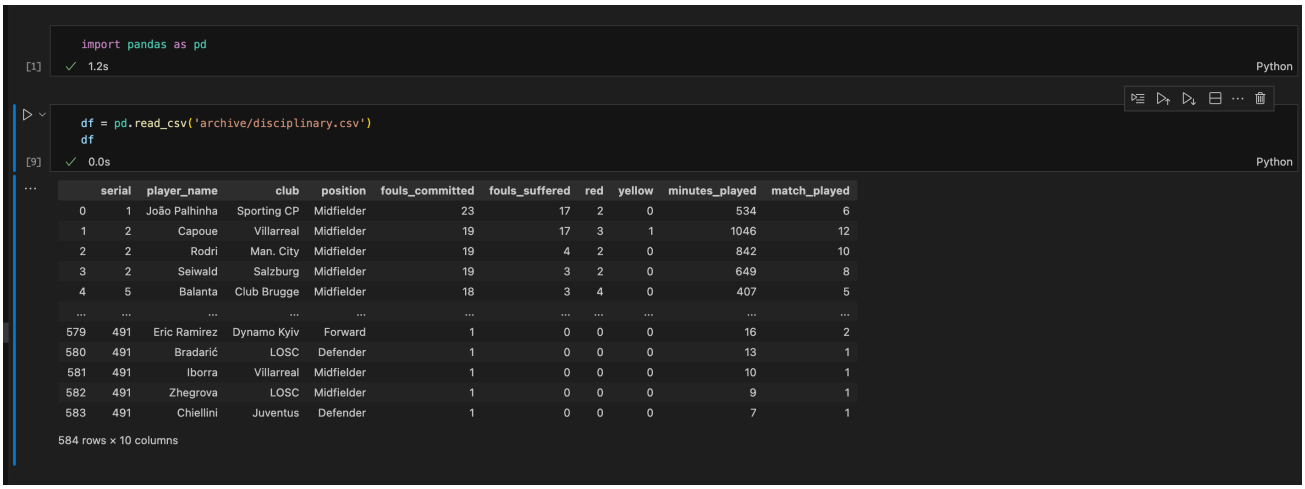
```
df = pd.read_csv('archive/defending.csv')
df
```
[8] ✓ 0.0s Python

	serial	player_name	club	position	balls_recoverd	tackles	t_won	t_lost	clearance_attempted	match_played
0	1	Casemiro	Real Madrid	Midfielder	76	31	13	18	19	11
1	1	Éder Militão	Real Madrid	Defender	76	19	6	13	32	12
2	1	Reinildo	Atlético	Defender	76	13	6	7	20	10
3	4	Fabinho	Liverpool	Midfielder	74	23	9	14	8	13
4	5	Albiol	Villarreal	Defender	73	9	6	3	62	12
...	...	...	...	...	...	...	...	...	...	...
622	578	Maier	Young Boys	Midfielder	1	0	0	0	0	1
623	578	Zhegrova	LOSC	Midfielder	1	0	0	0	0	1
624	578	Zesiger	Young Boys	Defender	1	0	0	0	0	1
625	578	Zakaria	Juventus	Midfielder	1	0	0	0	0	1
626	578	Edwards	Sporting CP	Forward	1	0	0	0	0	1

627 rows × 10 columns

The next csv file is called defending.csv and as we can see, this file contains 10 columns and 627 rows with stats like player name, club, position, balls recovered, tackles, tackles won, tackles lost, attempted clearances and matches played. This is a csv file that contains more defense related statistics and is used to analyse defenders and deep lying midfielders

4. DISCIPLINARY.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

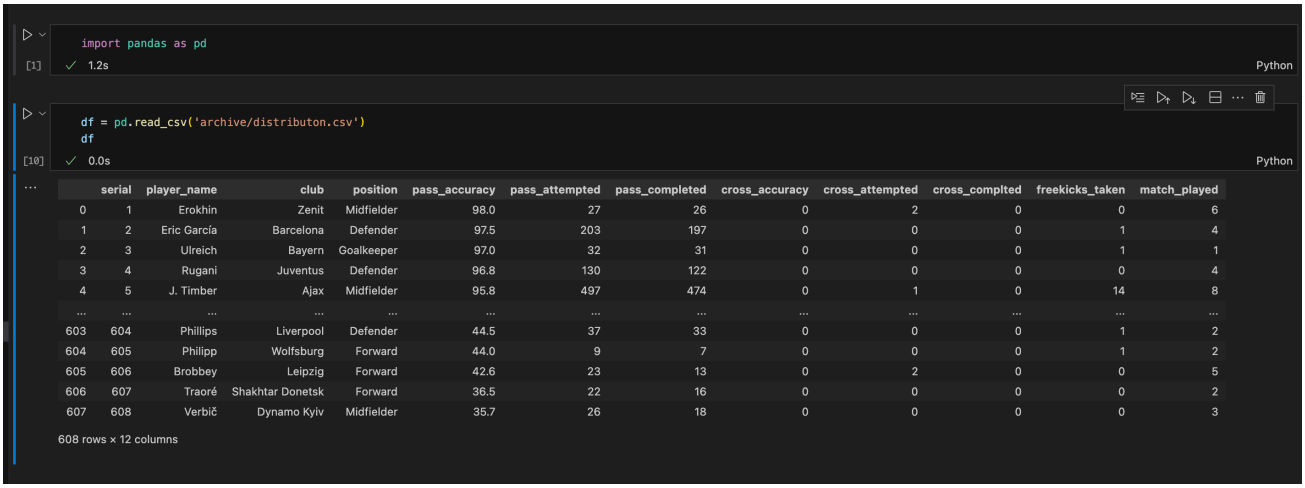
```
df = pd.read_csv('archive/disciplinary.csv')
df
```
[9] ✓ 0.0s Python

	serial	player_name	club	position	fouls_committed	fouls_suffered	red	yellow	minutes_played	match_played
0	1	João Palhinha	Sporting CP	Midfielder	23	17	2	0	534	6
1	2	Capoue	Villarreal	Midfielder	19	17	3	1	1046	12
2	2	Rodri	Man. City	Midfielder	19	4	2	0	842	10
3	2	Seiwald	Salzburg	Midfielder	19	3	2	0	649	8
4	5	Balanta	Club Brugge	Midfielder	18	3	4	0	407	5
...	...	...	...	...	...	...	...	...	...	...
579	491	Eric Ramirez	Dynamo Kyiv	Forward	1	0	0	0	16	2
580	491	Bradarić	LOSC	Defender	1	0	0	0	13	1
581	491	Iborra	Villarreal	Midfielder	1	0	0	0	10	1
582	491	Zhegrova	LOSC	Midfielder	1	0	0	0	9	1
583	491	Chiellini	Juventus	Defender	1	0	0	0	7	1

584 rows × 10 columns

The next csv file is called disciplinary.csv and as we can see, this file contains 10 columns and 584 rows with stats like player name, club, position, fouls committed, fouls suffered, red cards, yellow cards, minutes played and matches played. This is a csv file that contains stats which help us analyse how clean a player is on the field without having to take on cards or giving away fouls. It also helps us analyse how much certain players have been targeted. Its primarily used for midfield players

5. DISTRIBUTION.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

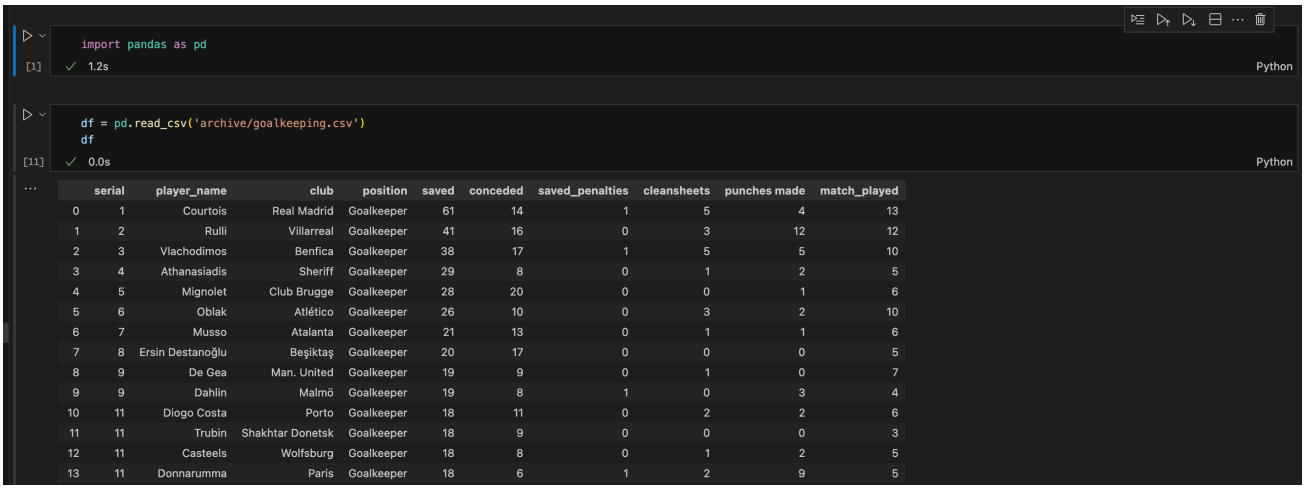
```
df = pd.read_csv('archive/distributon.csv')
df
```
[10] ✓ 0.0s Python

	serial	player_name	club	position	pass_accuracy	pass_attempted	pass_completed	cross_accuracy	cross_attempted	cross_complted	freekicks_taken	match_played
0	1	Erokhin	Zenit	Midfielder	98.0	27	26	0	2	0	0	6
1	2	Eric García	Barcelona	Defender	97.5	203	197	0	0	0	1	4
2	3	Ulreich	Bayern	Goalkeeper	97.0	32	31	0	0	0	1	1
3	4	Rugani	Juventus	Defender	96.8	130	122	0	0	0	0	4
4	5	J. Timber	Ajax	Midfielder	95.8	497	474	0	1	0	14	8
...	...	...	...	...	...	...	...	...	...	...	...	...
603	604	Phillips	Liverpool	Defender	44.5	37	33	0	0	0	1	2
604	605	Philipp	Wolfsburg	Forward	44.0	9	7	0	0	0	1	2
605	606	Brobbey	Leipzig	Forward	42.6	23	13	0	2	0	0	5
606	607	Traoré	Shakhtar Donetsk	Forward	36.5	22	16	0	0	0	0	2
607	608	Verbič	Dynamo Kyiv	Midfielder	35.7	26	18	0	0	0	0	3

608 rows × 12 columns

The next csv file is called distribution.csv as we can see, this file contains 12 columns and 608 rows with stats like player name, club, position, pass accuracy, passes completed , passes attempted, cross accuracy, cross attempted, crosses completed, freekicks taken and matches played. This is a csv file that contains stats which help us analyse how well a player can pass a ball. It also helps us analyse how well a player can cross a ball which is a very important aspect of footballing attack

6. GOALKEEPING.CSV



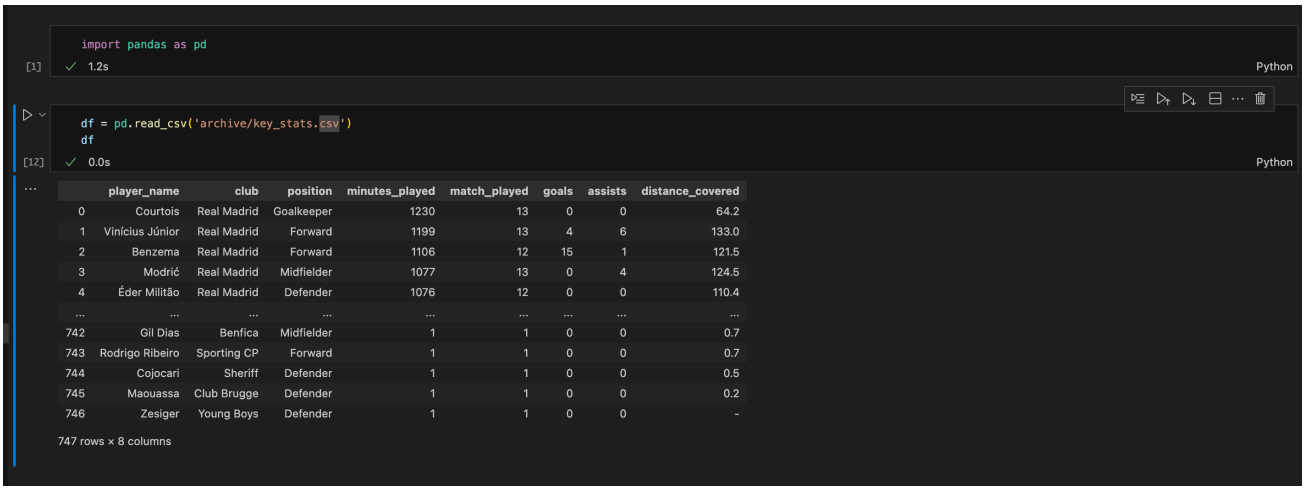
```
import pandas as pd
```
[1] ✓ 1.2s Python

```
df = pd.read_csv('archive/goalkeeping.csv')
df
```
[11] ✓ 0.0s Python

	serial	player_name	club	position	saved	conceded	saved_penalties	cleansheets	punches made	match_played
0	1	Courtois	Real Madrid	Goalkeeper	61	14	1	5	4	13
1	2	Rulli	Villarreal	Goalkeeper	41	16	0	3	12	12
2	3	Vlachodimos	Benfica	Goalkeeper	38	17	1	5	5	10
3	4	Athanasiadis	Sheriff	Goalkeeper	29	8	0	1	2	5
4	5	Mignolet	Club Brugge	Goalkeeper	28	20	0	0	1	6
5	6	Oblak	Atlético	Goalkeeper	26	10	0	3	2	10
6	7	Musso	Atalanta	Goalkeeper	21	13	0	1	1	6
7	8	Ersin Destanoğlu	Beşiktaş	Goalkeeper	20	17	0	0	0	5
8	9	De Gea	Man. United	Goalkeeper	19	9	0	1	0	7
9	9	Dahlin	Malmö	Goalkeeper	19	8	1	0	3	4
10	11	Diogo Costa	Porto	Goalkeeper	18	11	0	2	2	6
11	11	Trubin	Shakhtar Donetsk	Goalkeeper	18	9	0	0	0	3
12	11	Casteels	Wolfsburg	Goalkeeper	18	8	0	1	2	5
13	11	Donnarumma	Paris	Goalkeeper	18	6	1	2	9	5

The next csv file is called goalkeeping.csv as we can see, this file contains 10 columns and 52 rows with the stats of all the goalkeepers in the league like Player name, club, position, shots saved, shots conceded, saved penalties, clean sheets, punches made, and matches played. This is a csv file that contains stats which help us analyse how well a goalkeeper has performed over the course of the season

7. KEY_STATS.CSV



```
import pandas as pd
```
[1] ✓ 1.2s Python

```
df = pd.read_csv('archive/key_stats.csv')
df
```
[12] ✓ 0.0s Python

	player_name	club	position	minutes_played	match_played	goals	assists	distance_covered
0	Courtois	Real Madrid	Goalkeeper	1230	13	0	0	64.2
1	Vinícius Júnior	Real Madrid	Forward	1199	13	4	6	133.0
2	Benzema	Real Madrid	Forward	1106	12	15	1	121.5
3	Modrić	Real Madrid	Midfielder	1077	13	0	4	124.5
4	Éder Militão	Real Madrid	Defender	1076	12	0	0	110.4
...	...	...	...	...	...	...	...	...
742	Gil Dias	Benfica	Midfielder	1	1	0	0	0.7
743	Rodrigo Ribeiro	Sporting CP	Forward	1	1	0	0	0.7
744	Cojocari	Sheriff	Defender	1	1	0	0	0.5
745	Maouassa	Club Brugge	Defender	1	1	0	0	0.2
746	Zesiger	Young Boys	Defender	1	1	0	0	-

747 rows × 8 columns

The next csv file is called key_stats.csv as we can see, this file contains 8 columns and 747 rows with stats like Player name, club position, minutes played, matches played, goals, assist and distance covered. This is a csv file that contains stats which help us analyse how much certain players run over the course of the season along with their goal and assist contribution

8. GOALS.CSV

```
import pandas as pd
```
[1] ✓ 1.2s Python

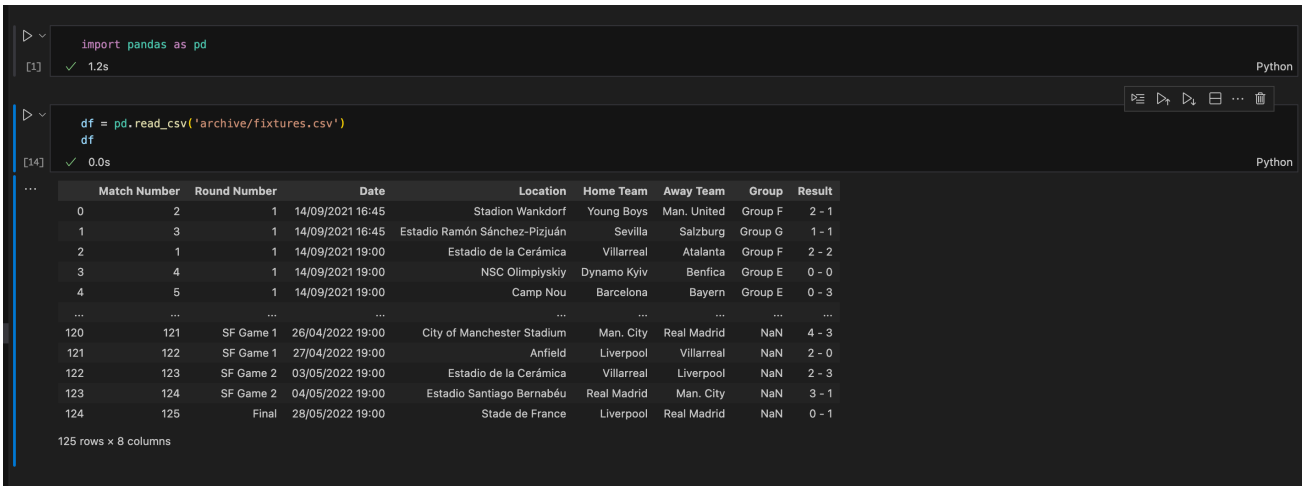
```
df = pd.read_csv('archive/goals.csv')
df
```
[13] ✓ 0.0s Python

	serial	player_name	club	position	goals	right_foot	left_foot	headers	others	inside_area	outside_areas	penalties	match_played
0	1	Benzema	Real Madrid	Forward	15	11	1	3	0	13	2	3	12
1	2	Lewandowski	Bayern	Forward	13	8	3	1	1	13	0	3	10
2	3	Haller	Ajax	Forward	11	3	4	3	1	11	0	1	8
3	4	Salah	Liverpool	Forward	8	0	8	0	0	7	1	1	13
4	5	Nkunku	Leipzig	Midfielder	7	3	1	3	0	7	0	0	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
178	73	Yansané	Sheriff	Forward	1	0	0	1	0	1	0	0	3
179	73	Yakhshiboev	Sheriff	Forward	1	0	0	1	0	1	0	0	3
180	73	Messias Junior	Milan	Forward	1	0	0	1	0	1	0	0	2
181	73	Morato	Benfica	Defender	1	0	0	1	0	1	0	0	2
182	73	Pedro Porro	Sporting CP	Defender	1	0	0	0	1	1	0	0	7

183 rows × 13 columns

The next csv file is called goals.csv as we can see, this file contains 13 columns and 183 rows with stats of all the goalscorers in the season like Player name, club, position, goals scored, goals scored with the right foot, goals scored with the left foot, headers scored, other parts of the body used to score, goals scored from inside the penalty area, goals scored from outside the penalty area, penalties scored and matches played. This is a csv file that contains stats which help us analyse the different types of goals scored and the different goals scored by the players

9. FIXTURES.CSV



```
import pandas as pd
```

[1] ✓ 1.2s Python

```
df = pd.read_csv('archive/fixtures.csv')
df
```

[14] ✓ 0.0s Python

	Match Number	Round Number	Date	Location	Home Team	Away Team	Group	Result
0	2	1	14/09/2021 16:45	Stadion Wankdorf	Young Boys	Man. United	Group F	2 - 1
1	3	1	14/09/2021 16:45	Estadio Ramón Sánchez-Pizjuán	Sevilla	Salzburg	Group G	1 - 1
2	1	1	14/09/2021 19:00	Estadio de la Cerámica	Villarreal	Atalanta	Group F	2 - 2
3	4	1	14/09/2021 19:00	NSC Olimpiyskiy	Dynamo Kyiv	Benfica	Group E	0 - 0
4	5	1	14/09/2021 19:00	Camp Nou	Barcelona	Bayern	Group E	0 - 3
...	...	...	...	...	...	...	...	...
120	121	SF Game 1	26/04/2022 19:00	City of Manchester Stadium	Man. City	Real Madrid	NaN	4 - 3
121	122	SF Game 1	27/04/2022 19:00	Anfield	Liverpool	Villarreal	NaN	2 - 0
122	123	SF Game 2	03/05/2022 19:00	Estadio de la Cerámica	Villarreal	Liverpool	NaN	2 - 3
123	124	SF Game 2	04/05/2022 19:00	Estadio Santiago Bernabéu	Real Madrid	Man. City	NaN	3 - 1
124	125	Final	28/05/2022 19:00	Stade de France	Liverpool	Real Madrid	NaN	0 - 1

125 rows × 8 columns

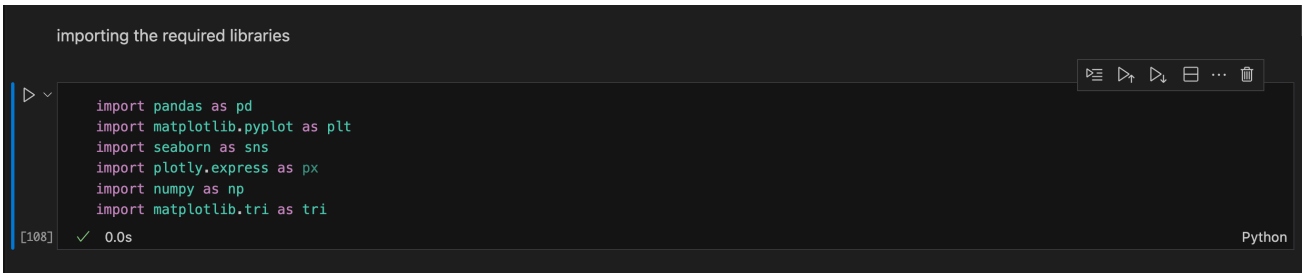
The next CSV file is the fixtures.CSV, and has 125 rows and 9 columns. By analyzing this, we can see all the different games played through the course of the season along with the time, the date, the location, the home team, the away team, the group in which they played and the result of the game played

in order to have more effective analyzation, I have divided all these data sets in to four different python notebooks and each of these notebooks are used separately to analyze different data and different statistics for different players based on their separate positions

ATTACKINGSTATS.IPYNB

the first notebook is the attackingStats notebook and I have used it in order to analyse the attacking output of the forward players in the league. These include the forwards, the attacking midfielders and so on

IMPORTING NECESSARY LIBRARIES



Pandas (imported as **pd**):

Pandas is a powerful data manipulation and analysis library for Python. It provides data structures such as DataFrame and Series, which allow for easy handling and manipulation of structured data. Pandas is commonly used for tasks such as data cleaning, transformation, exploration, and analysis.

Matplotlib (imported as **plt**):

Matplotlib is a popular plotting library for Python. It provides a wide variety of functions and methods for creating static, interactive, and animated visualizations. Matplotlib can be used to generate plots, histograms, scatter plots, bar charts, line charts, and more, making it a versatile tool for data visualization.

Seaborn (imported as **sns**):

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations such as heatmaps, violin plots, pair plots, and regression plots, with options for styling and customization.

Plotly Express (imported as **px**)

Plotly Express is a high-level interface for creating interactive visualizations using Plotly, a leading visualization library for Python. Plotly Express simplifies the creation of

interactive plots such as scatter plots, line plots, bar charts, and 3D plots, with built-in support for interactivity, animation, and customization.

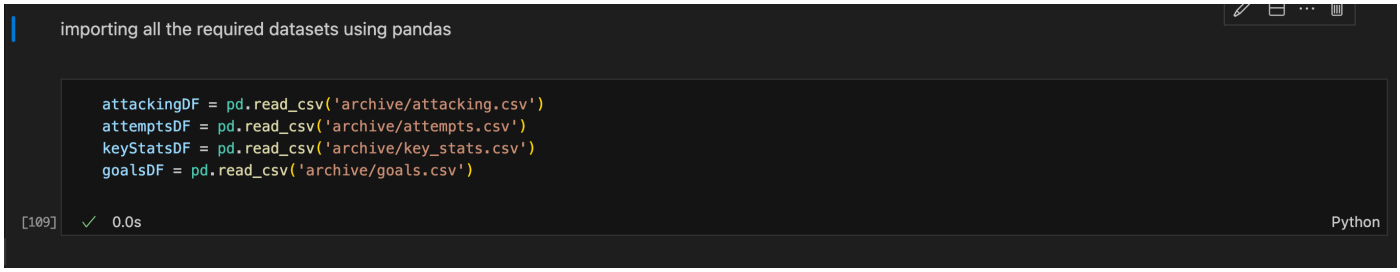
NumPy (imported as **np**)

NumPy is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays, mathematical functions, linear algebra operations, random number generation, and more. NumPy is widely used for numerical analysis, scientific computing, and data manipulation tasks.

Matplotlib Triangular Mesh (imported as **tri**):

Matplotlib Triangular Mesh is a module within Matplotlib that provides functionality for creating and plotting triangular meshes. It allows for the creation of plots and visualizations based on triangular mesh data structures, which are commonly used in fields such as finite element analysis, computational fluid dynamics, and geospatial analysis.

IMPORTING REQUIRED DATASETS



read_csv(): This is a function provided by the Pandas library for reading data from CSV files. It's called using the dot notation (**pd.read_csv()**), where **pd** is the alias for the Pandas library.

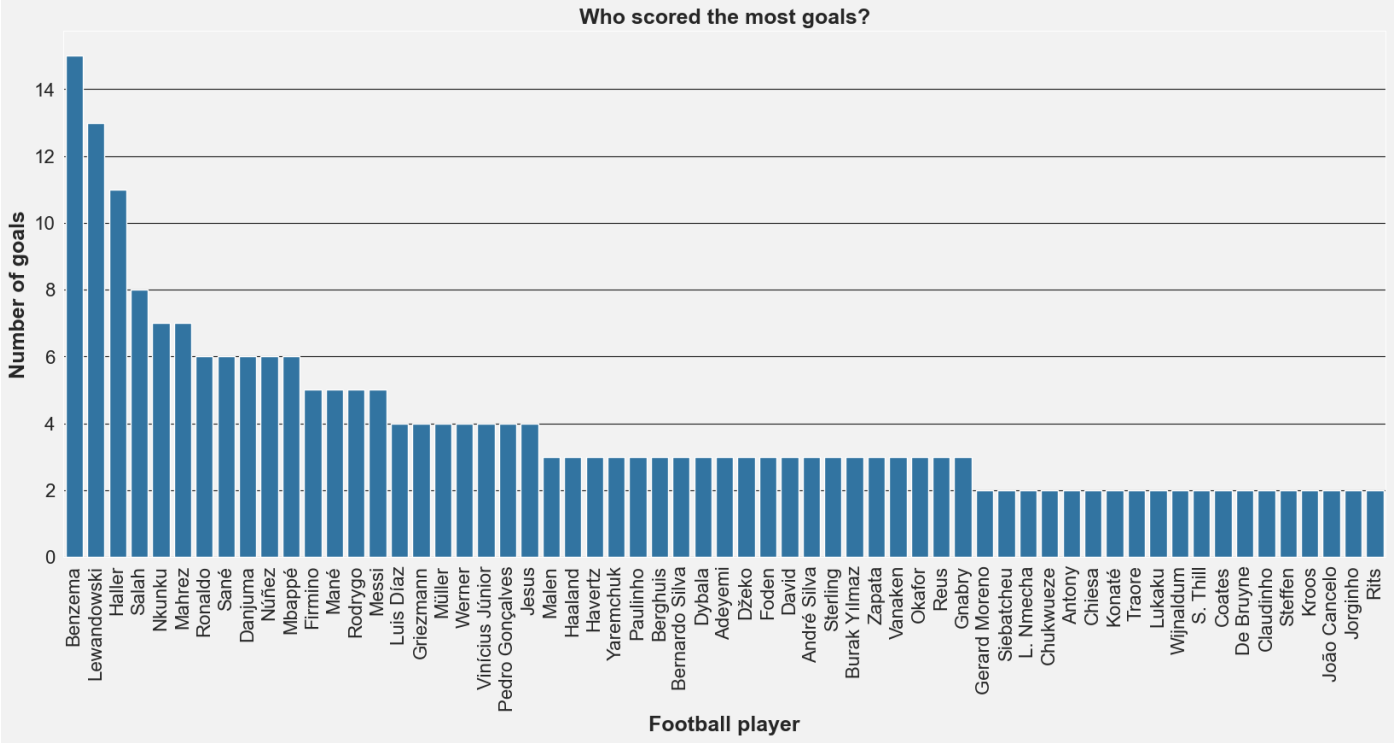
PLOTTING A BAR CHART FOR THE PLAYERS WITH THE MOST GOALS

bar chart showing players with the most goals

```
df_players = goalsDF.loc[0:41, ['player_name', 'goals']]

plt.figure(figsize=(20, 8))

plt.title('Who scored the most goals?', fontsize=18, fontweight='bold')
plt.xticks(fontsize=16, rotation=90)
plt.yticks(fontsize=16)

sns.set_style('darkgrid',
              {'axes.facecolor': '0.95',
               'grid.color': '0.1',
               'figure.facecolor': '0.95'})

sns.barplot(x=goalsDF.sort_values('goals', ascending=False)['player_name'][0:61],
            y=goalsDF.sort_values('goals', ascending=False)['goals'][0:61])

plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Number of goals', fontsize=18, fontweight='bold')
plt.show()
```

[110] ✓ 1.5s Python

OUTPUT



From this graph, we can infer that benzema and Lewandowski have scored the most goals in the league for the given season, and we can also figure out who the other top 61 players with the most goals are

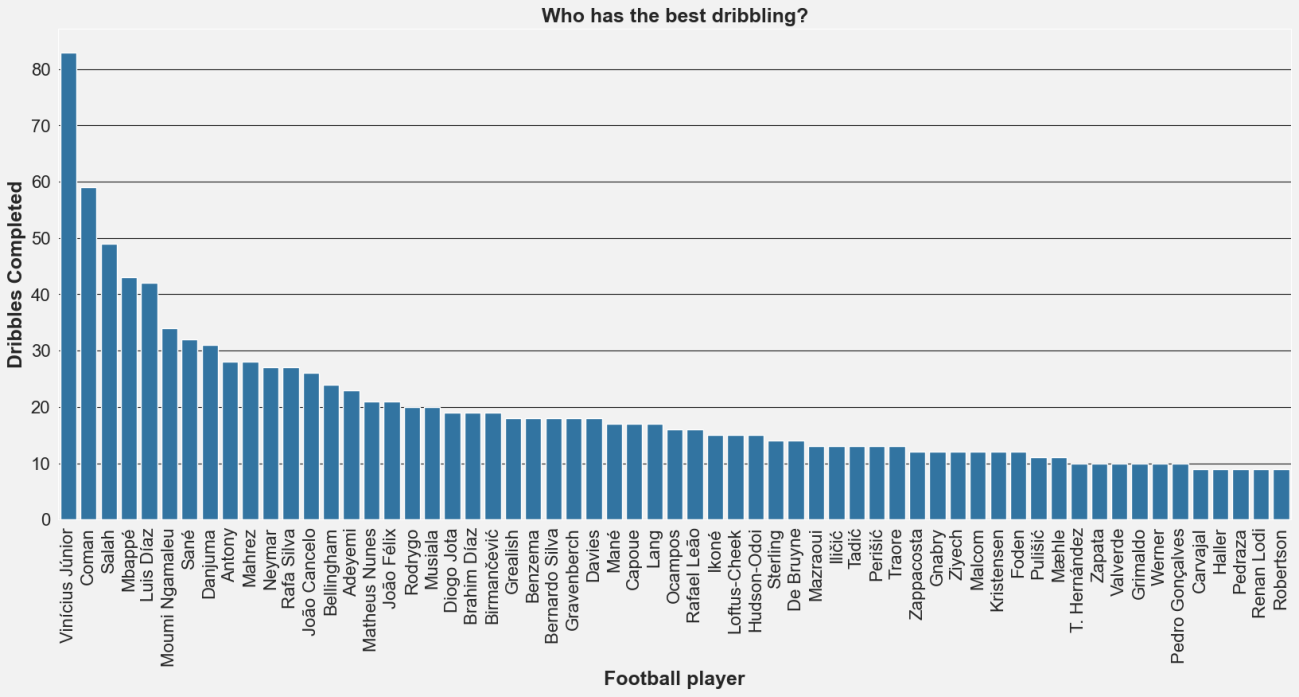
PLAYERS WITH THE MOST DRIBBLES COMPLETED IN THE GIVEN SEASON

bar graph to show the best dribblers in the league

```
plt.figure(figsize=(20, 8))

plt.title('Who has the best dribbling?', fontsize=18, fontweight='bold')

plt.xticks(rotation=90, fontsize=16)
plt.yticks(fontsize=16)

sns.set_style('darkgrid',
              {'axes.facecolor': '0.95',
               'grid.color': '0.1',
               'figure.facecolor': '0.95'})

sns.barplot(x=attackingDF.sort_values('dribbles', ascending=False)[0:61]['player_name'],
            y=attackingDF.sort_values('dribbles', ascending=False)[0:61]['dribbles'])

plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Dribbles Completed', fontsize=18, fontweight='bold')

plt.show()
```

[111] ✓ 2.3s Python

OUTPUT



From this graph we can infer that Vinicius Juniór has completed the most dribbles in season, and we can also find out who the top 61 players with the most dribbles for the season are

RATIO OF ATTEMPTS ON AND OFF TARGET FOR A PARTICULAR PLAYER

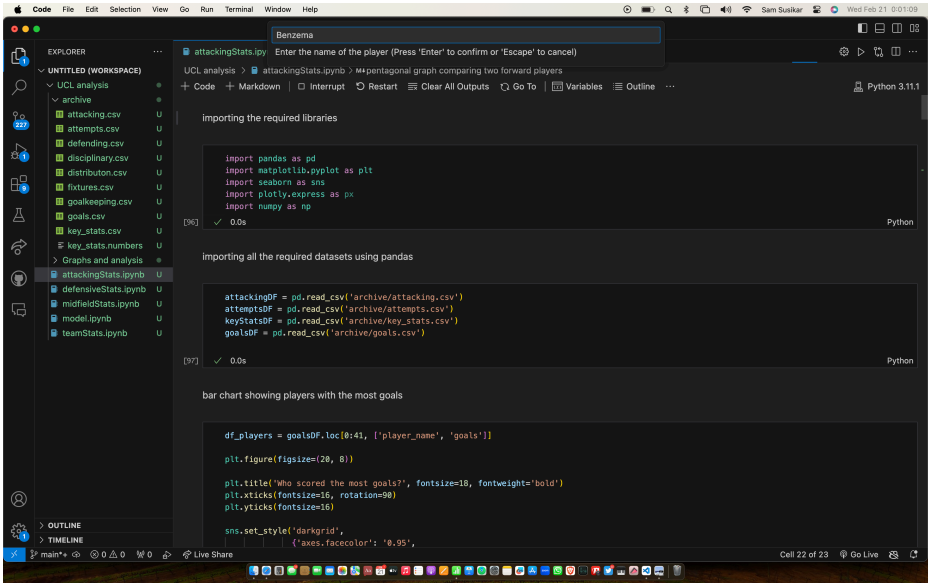
pie chart for attempts on target, off target and blocked

```
playerName = input("Enter the name of the player")
playerName2 = input("Enter the name of the player")
playerStats = attemptsDF[attemptsDF['player_name'] == playerName]
onTarget = playerStats['on_target'].iloc[0]
offTarget = playerStats['off_target'].iloc[0]
blocked = playerStats['blocked'].iloc[0]
plt.title(playerName+" shooting stats", fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})
plt.pie([onTarget, offTarget, blocked],labels=['On target', 'Off target', 'Blocked'], colors=['#013369','#000000','#333333'])
plt.legend()

playerStats
```

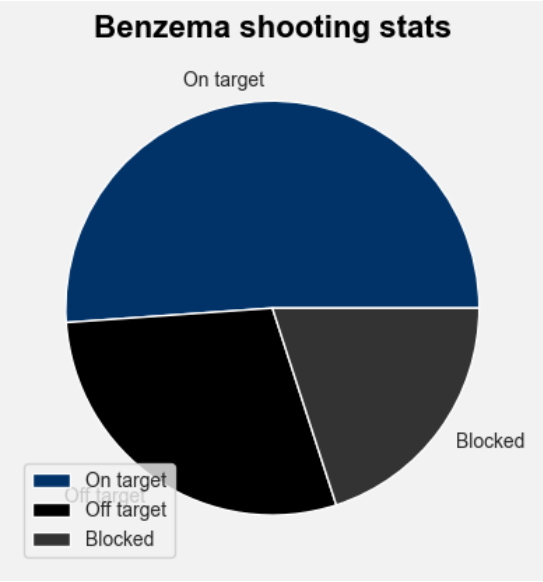
[203] ✓ 4.9s Python

	serial	player_name	club	position	total_attempts	on_target	off_target	blocked	match_played
0	1	Benzema	Real Madrid	Forward	45	23	13	9	12

OUTPUT



User input for the player name



shooting stats for player one

SCATTER PLOT FOR THE MATCHES PLAYED VS ATTEMPTS TAKEN FOR ALL THE FORWARD PLAYERS

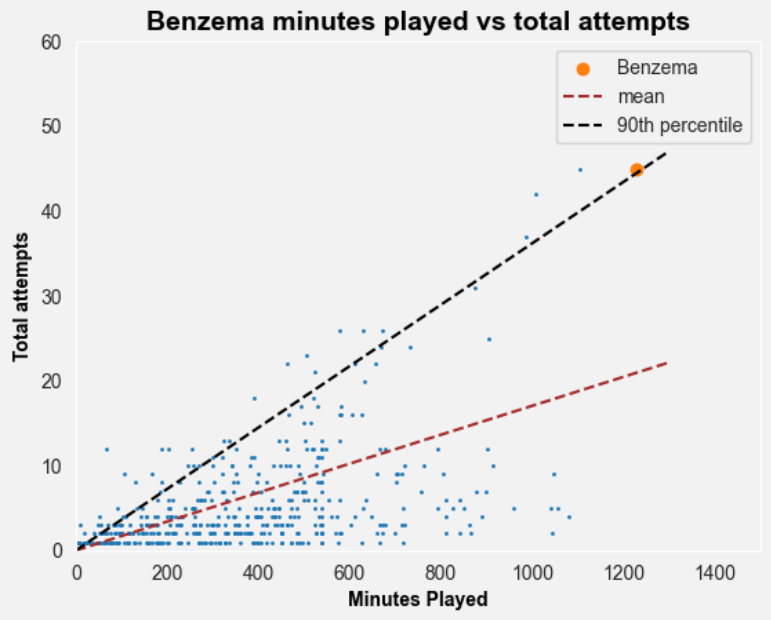
scatter plot for matches played vs attempts

+ Code + Markdown

```
# x and y limits based on the given dataset
plt.ylim(0,60)
plt.xlim(0,1500)


plt.xlabel("Minutes Played", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Total attempts", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})

# merging minutes played and attempts from attemptsDF and keyStatsDF
keyXattempts = pd.merge(attemptsDF, keyStatsDF, on='player_name')

# scatter plot 1 for every player in the league
plt.scatter(keyXattempts['minutes_played'], keyXattempts['total_attempts'],s=1)
minutesPlayed = keyStatsDF['minutes_played'].iloc[0]
totalAttempts = playerStats['total_attempts'].iloc[0]

# scatter plot 2 for given searched player
plt.scatter(minutesPlayed, totalAttempts, label=playerName)
plt.title(playerName+" minutes played vs total attempts", fontdict={'fontname': 'Arial', 'fontsize': 14, 'fontweight': 'bold', 'color': 'black'})



# Mean line
average_y = np.mean(attemptsDF['total_attempts'])
average_x = np.mean(keyStatsDF['minutes_played'])
slope=(average_y)/(average_x)
x_values = np.linspace(0, 1300, 15000)
y_values = slope*(x_values)
plt.plot(x_values, y_values, color='brown',linestyle='--',label='mean')


# 90th percentile line
percentile90 =np.percentile(attemptsDF['total_attempts'], 90)
slope90 = percentile90/average_x
x_values90 = np.linspace(0, 1300, 15000)
y_values90 = slope90*(x_values90)
plt.plot(x_values90, y_values90, color='black',linestyle='--',label='90th percentile')

plt.grid(False)

plt.legend()
```

[113] ✓ 0.5s Python

OUTPUT



This is a Scatterplot for all the forward players in the league, the black line represents the 90th percentile of players and the brown line is the average line. The orange dot represents the player in question, here, Karim Benzema

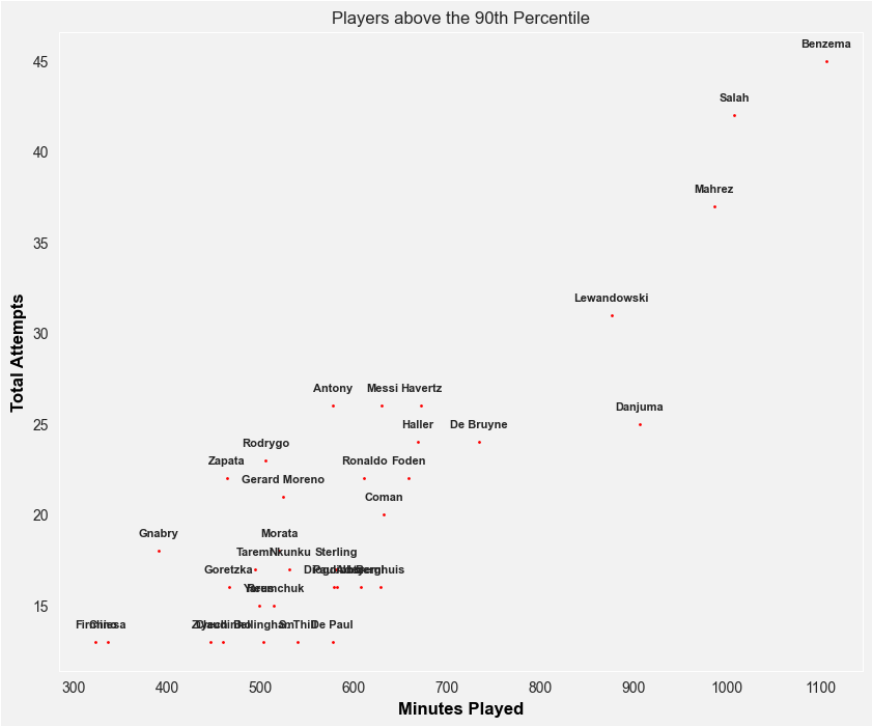
EXTRACTING THE PLAYERS ABOVE THE 90TH PERCENTILE

extracting the top players above the 90th percentile

```
# Setting the figure size
plt.figure(figsize=(10, 8))


plt.xlabel("Minutes Played", fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Total attempts", fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Iterate through keyXattempts and identify players above the 90th percentile line
for index, row in keyXattempts.iterrows():
    minutes_played = row['minutes_played']
    total_attempts = row['total_attempts']

    # Get the player's name from the current row
    player_name = row['player_name']

    # Check if total_attempts is above the 90th percentile
    if total_attempts > percentile90:
        # Plot a dot for the player above the 90th percentile line
        plt.scatter(minutes_played, total_attempts, color='red', label=player_name, s=10)

        # Annotate the player's stats
        plt.annotate(player_name, (minutes_played, total_attempts), textcoords="offset points", xytext=(0,10), ha='center', fontsize=8, fontweight='bold')

        plt.grid(False)

# Filter players above the 90th percentile line
players_above_90th = keyXattempts[keyXattempts['total_attempts'] > percentile90]

# Displaying their names in a DataFrame
players_df = players_above_90th[['player_name', 'minutes_played', 'total_attempts']]
print(players_df)


plt.xlabel("Minutes Played", fontdict={'fontname': 'Arial', 'fontsize': 12, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Total Attempts", fontdict={'fontname': 'Arial', 'fontsize': 12, 'fontweight': 'bold', 'color': 'black'})

plt.title("Players above the 90th Percentile")


plt.show()
```

[114] ✓ 0.9s Python

OUTPUT



Scatter plot above the 90th percentile

...	player_name	minutes_played	total_attempts
0	Benzema	1106	45
1	Salah	1008	42
2	Mahrez	986	37
3	Lewandowski	876	31
4	Havertz	672	26
5	Antony	577	26
6	Messi	630	26
7	Danjuma	906	25
8	Haller	668	24
9	De Bruyne	734	24
10	Rodrygo	505	23
11	Ronaldo	611	22
12	Foden	658	22
13	Zapata	464	22
14	Gerard Moreno	524	21
15	Coman	632	20
16	Gnabry	391	18
17	Morata	520	18
18	Nkunku	531	17
19	Sterling	581	17
20	Taremi	494	17
21	Adeyemi	607	16
22	Paulinho	582	16
23	Diogo Jota	578	16
...			
31	Chiesa	336	13
32	De Paul	577	13
33	Claudinho	460	13
34	Bellingham	503	13

Players above the 90th percentile

SCATTER PLOT FOR CONVERSION RATE

conversion rate

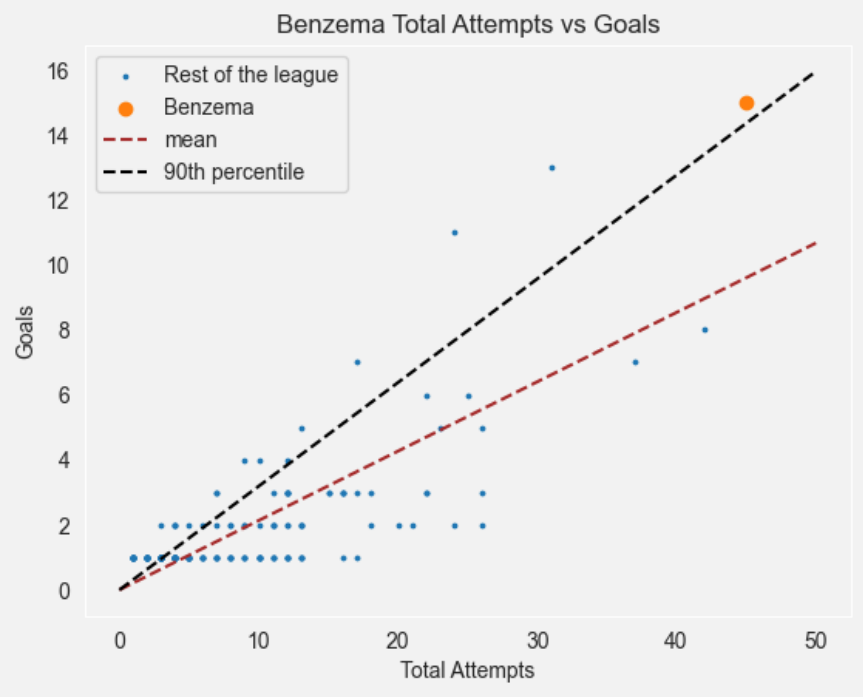
```
# importing goal stats dataset
goalsDF = pd.read_csv('archive/goals.csv')

# merging the two dataframes values based on the common denomiator (player name)
combined_df = pd.merge(attemptsDF, goalsDF, on='player_name')

# scatter plot for the rest of the league
plt.scatter(combined_df['total_attempts'], combined_df['goals'], s=3, label="Rest of the league")
plt.xlabel('Total Attempts')
plt.ylabel('Goals')
plt.title(playerName+' Total Attempts vs Goals')

# scatter plot for a given player
playerGoal = combined_df[combined_df['player_name'] == playerName]
plt.scatter(playerGoal['total_attempts'],playerGoal['goals'],label=playerName)


# mean line
average_y = np.mean(combined_df['goals'])
average_x = np.mean(combined_df['total_attempts'])
slope=(average_y)/(average_x)
x_values = np.linspace(0, 50, 1000)
y_values = slope*(x_values)
plt.plot(x_values, y_values, color='brown',linestyle='--',label='mean')

# 90th percentile line
percentile90 =np.percentile(combined_df['goals'], 90)
slope90 = percentile90/average_x
x_values90 = np.linspace(0, 50, 1000)
y_values90 = slope90*(x_values90)
plt.plot(x_values90, y_values90, color='black',linestyle='--',label='90th percentile')

plt.legend()

plt.grid(False)

combined_df[combined_df['player_name'] == playerName]
```

[115] ✓ 0.5s Python

	serial_x	player_name	club_x	position_x	total_attempts	on_target	off_target	blocked	match_played_x	serial_y	...	position_y	goals	right_foot	left_foot	headers	others	inside_area	outside_areas	penalties	match_played_y
3	5	Lewandowski	Bayern	Forward	31	19	8	4	10	2	...	Forward	13	8	3	1	1	13	0	3	10

1 rows × 21 columns

OUTPUT



Scatter plot for the total attempts vs the goals scored, the orange dot represents the player in question

CUSTOM MADE RADAR CHART

This custom radar chart is a visualization tool used to compare multiple variables, here, various player statistics (Total attempts per 90 (5), Goals per 90 (2), Shots on target per 90 (4), Dribbles per 90 (2), Assists per 90) simultaneously. Here's a breakdown of its components and their relevance:

Data Representation: Each variable is represented by a axis extending from the center of the chart to its circumference. The values of each variable are represented by the distance from the center. This allows for easy comparison between different variables for each player.

Labels: Each axis is labeled with the corresponding player statistic, making it easy to understand which axis represents which variable.

Color Scheme: The color scheme is used to differentiate between different variables, with each variable being assigned a unique color. This helps in visually identifying and distinguishing between different variables.

Background and Foreground Triangles: The background triangles provide context by filling the space between the axes, while the foreground triangles represent the actual data points. The alpha parameter controls the transparency of these triangles, allowing for better visualization of overlapping areas.

Title: The title provides context for the chart, indicating whose statistics are being compared.

Overall, this radar chart offers a visually appealing and informative way to compare player statistics across multiple variables, enabling quick insights into each player's performance in various aspects of the game. It's particularly useful when you want to analyze and compare players based on several key metrics simultaneously.

CODE TO EXTRACT THE STATS FOR PLAYER 1

radar chart stats for player 1

```
# Merge the data frames using the 'player_name' column
playerGraphDataDF = pd.merge(attemptsDF[['player_name', 'total_attempts', 'on_target', 'match_played']],
                             attackingDF[['player_name', 'assists', 'dribbles']],
                             on='player_name')
playerGraphDataDF = pd.merge(playerGraphDataDF,
                             goalsDF[['player_name', 'goals']],
                             on='player_name')
playerGraphDataDF = pd.merge(playerGraphDataDF,
                             keyStatsDF[['player_name', 'minutes_played']],
                             on='player_name')
player_row = playerGraphDataDF[playerGraphDataDF['player_name'] == playerName].iloc[0]


# extracting the minutes played and matches played
mp = player_row['match_played']
minp = player_row['minutes_played']

# listing all the stat to be plotted on the graph and storing them in a list
playerGraphStats = [(player_row['total_attempts'])*90/(minp*5), ((player_row['goals']*90))/(minp*2),
                    player_row['on_target']*90/(minp*4), player_row['dribbles']*90/(minp*2),
                    player_row['assists']*90/minp]


playerGraphStats
```

[116] ✓ 0.0s Python

```
[0.636986301369863,
 0.6678082191780822,
 0.488013698630137,
 0.05136986301369863,
 0.3082191780821918]
```

CODE TO EXTRACT THE STATS FOR PLAYER 2

radar chart stats for player 2

```
player_row2 = playerGraphDataDF[playerGraphDataDF['player_name'] == playerName2].iloc[0]

mp2 = player_row2['match_played']
minp2 = player_row2['minutes_played']

playerGraphStats2 = [(player_row2['total_attempts'])*90/(minp2*5), ((player_row2['goals']*90))/(minp2*2),
                    player_row2['on_target']*90/(minp2*4), player_row2['dribbles']*90/(minp2*2),
                    player_row2['assists']*90/minp2]

playerGraphStats2
```

[117] ✓ 0.0s Python

```
[0.7323688969258589,
 0.6103074141048824,
 0.4679023508137432,
 0.7323688969258589,
 0.08137432188065]
```

CODE FOR THE PENTAGONAL RADAR CHART

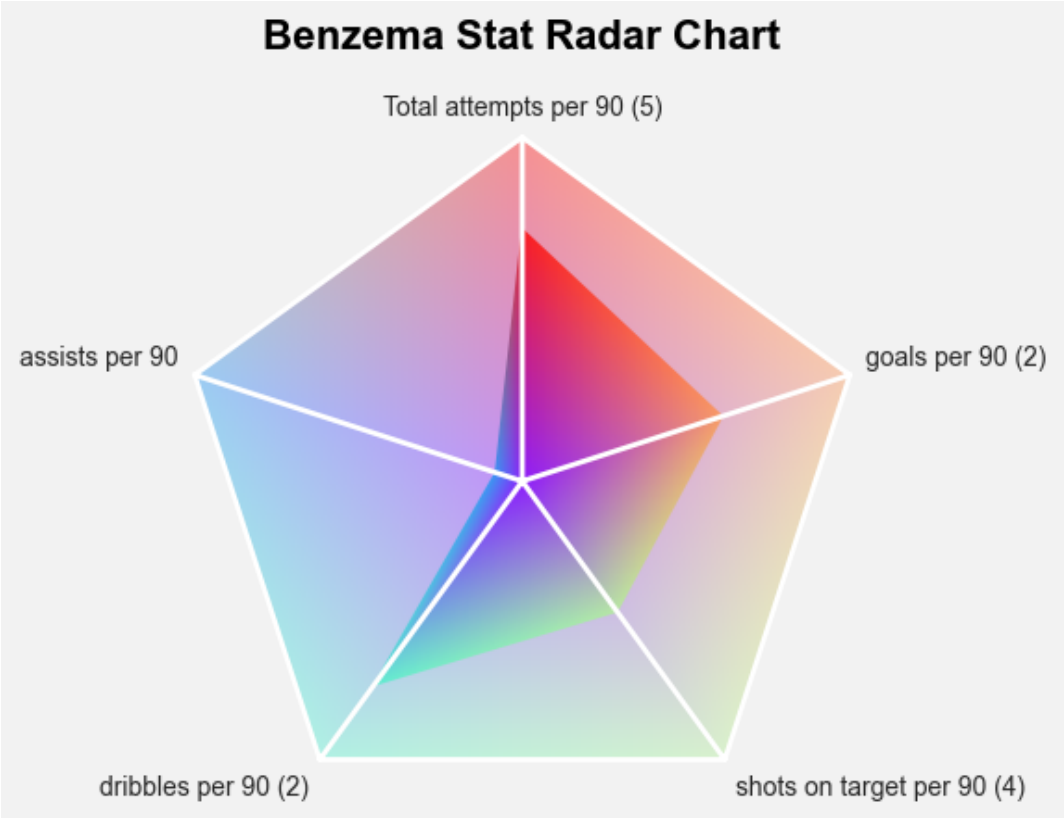
pentagonal graph with essential stats for a forward player

```
# Data for radar chart
proportions = playerGraphStats
labels = ['Total attempts per 90 (5)', 'Goals per 90 (2)',
          'Shots on target per 90 (4)', 'Dribbles per 90 (2)', 'Assists per 90']

# Number of variables
N = len(proportions)

# Add a point to close the radar chart
proportions = np.append(proportions, 1)

# Define angles for each axis
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)

# Define coordinates for the outer vertices of the radar chart
x = np.append(np.sin(theta), 0)
y = np.append(np.cos(theta), 0)

# Define triangles for background and foreground
triangles = [[N, i, (i + 1) % N] for i in range(N)]
triang_backgr = tri.Triangulation(x, y, triangles)
triang_foregr = tri.Triangulation(x * proportions, y * proportions, triangles)

# Define colormap
cmap = plt.cm.rainbow_r
colors = np.linspace(0, 1, N + 1)

# Plot background and foreground triangles
plt.tripcolor(triang_backgr, colors, cmap=cmap, shading='gouraud', alpha=0.4)
plt.tripcolor(triang_foregr, colors, cmap=cmap, shading='gouraud', alpha=0.8)

# Plot triangle edges
plt.triplot(triang_backgr, color='white', lw=2)

# Add labels for each axis
for label, color, xi, yi in zip(labels, colors, x, y):
    plt.text(xi * 1.05, yi * 1.05, label,
             ha='left' if xi > 0.1 else 'right' if xi < -0.1 else 'center',
             va='bottom' if yi > 0.1 else 'top' if yi < -0.1 else 'center')

# Turn off axis
plt.axis('off')

# Set aspect ratio to be equal
plt.gca().set_aspect('equal')

# Set title
plt.title(playerName + ' Stat Radar Chart\n', fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Show the plot
plt.show()
```

[118] ✓ 0.3s Python

OUTPUT FOR PLAYER 1

The



bracketed values are the limits defined and the shaded portion represents how well the player has done

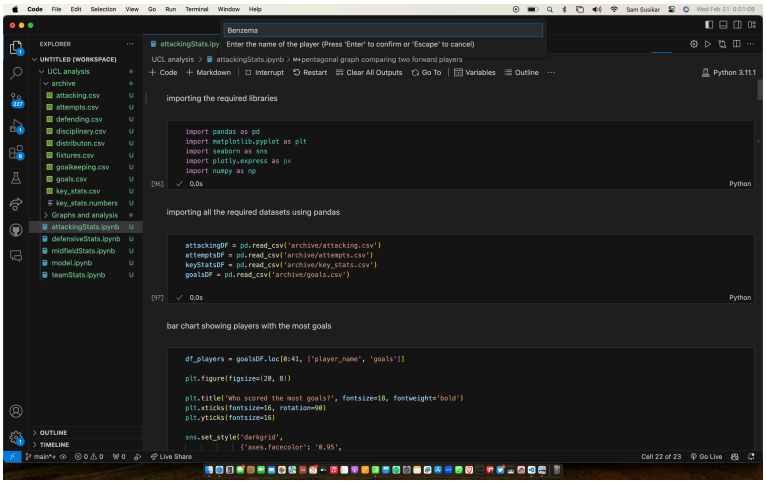
COMPAIRING PLAYER 1'S STATS WITH PLAYER 2

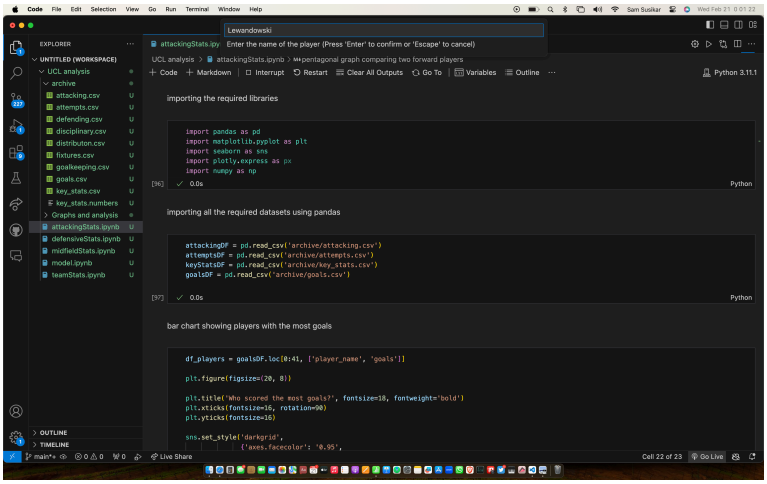
pentagonal graph comparing two forward players

```
# Assuming playerGraphStats1 and playerGraphStats2 are defined

proportions1 = playerGraphStats
proportions2 = playerGraphStats2

# Remove a label to match the length of proportions
labels = ['Total attempts per 90 (5)', 'goals per 90 (2)',
          'shots on target per 90 (4)', 'dribbles per 90 (2)', 'assists per 90']

# Define player statistics as strings
stats1 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions1)]
stats2 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions2)]

# Join the statistics into a single string with line breaks
stats_str1 = '\n'.join(stats1)
stats_str2 = '\n'.join(stats2)

N = len(labels)
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)
x = np.sin(theta)
y = np.cos(theta)

# Append the first vertex to the end to close the polygon
x = np.append(x, x[0])
y = np.append(y, y[0])

# Plotting the filled background pentagon with 5 equilateral triangles
for i in range(N):
    plt.plot([x[i], x[(i+1)%N]], [y[i], y[(i+1)%N]], color='lightgray', lw=2)

# Plotting the first player's statistics
plt.plot(x * np.append(proportions1, proportions1[0]), y * np.append(proportions1, proportions1[0]), color='blue', lw=2, label=playerName)

# Plotting the second player's statistics
plt.plot(x * np.append(proportions2, proportions2[0]), y * np.append(proportions2, proportions2[0]), color='red', lw=2, label=playerName2)

# Adding labels to the vertices
for label, xi, yi in zip(labels, x[:-1], y[:-1]):
    plt.text(xi * 1.05, yi * 1.05, label,
             ha='left' if xi > 0.1 else 'right' if xi < -0.1 else 'center',
             va='bottom' if yi > 0.1 else 'top' if yi < -0.1 else 'center')

# Adding legend and placing it on the extreme right
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Setting plot properties
plt.axis('off')
plt.gca().set_aspect('equal')
plt.title(playerName +' vs '+playerName2+'\n', fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Adding a text box with player statistics
plt.text(1.05, 0.5, stats_str1, bbox=dict(facecolor='lightblue', alpha=0.5), fontsize=8, verticalalignment='center')

plt.text(1.05, -0.5, stats_str2, bbox=dict(facecolor='lightcoral', alpha=0.5), fontsize=8, verticalalignment='center')

plt.show()
```

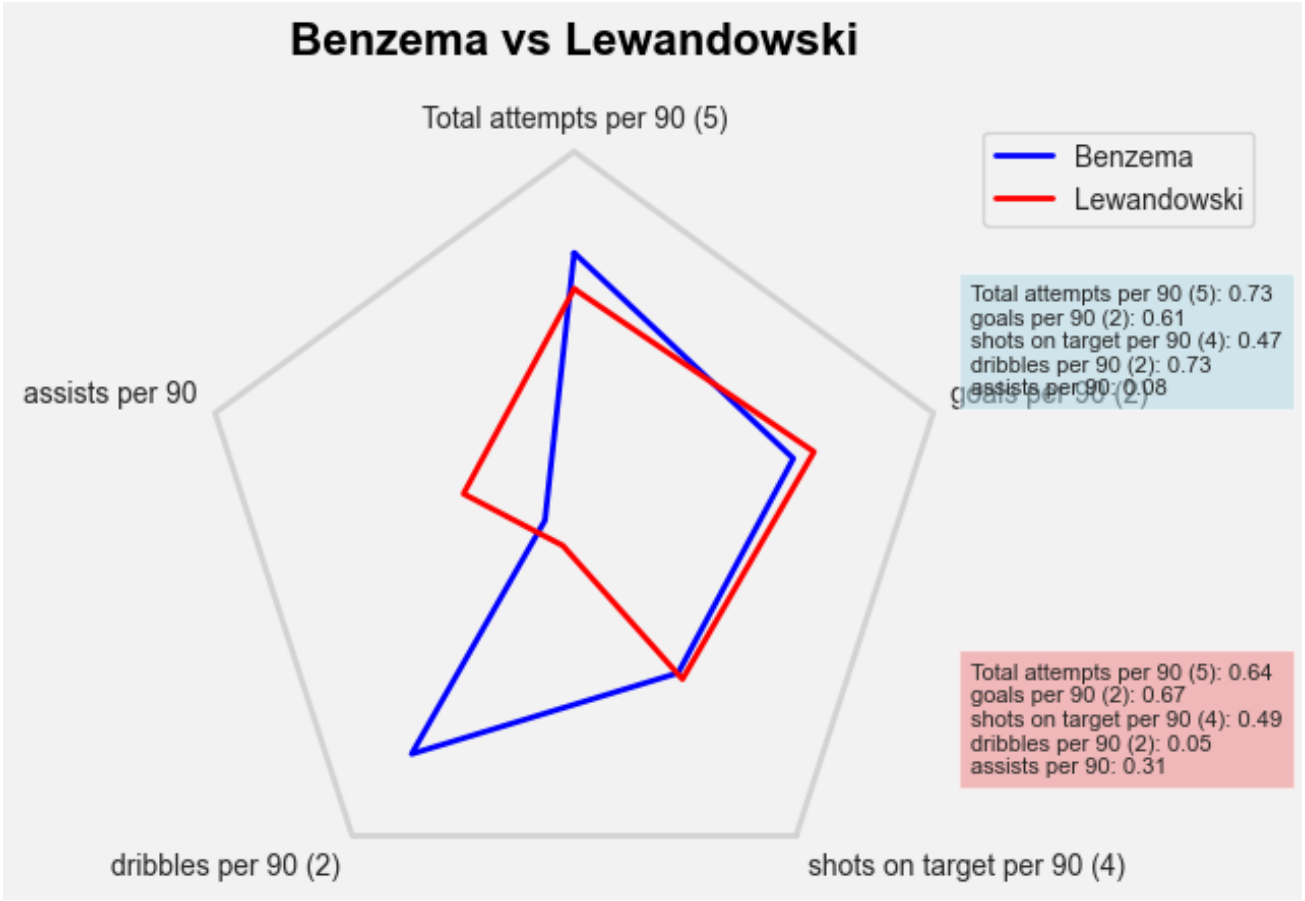
[119] ✓ 0.3s Python

TWO PLAYERS NAME AS INPUT





PLAYER ONE VS PLAYER TWO

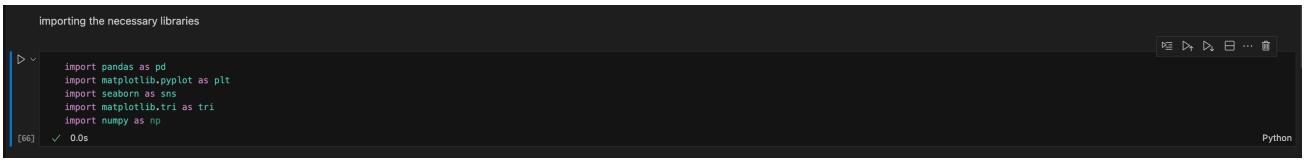


Radar chart of benzema plotted on top of the radar chart of lewandowski

MIDFIELDSTATS.IPYNB

the second notebook is the midfieldStats notebook and I have used it in order to analyse the attacking output of the midfield players in the league. These include the all the different midfielders (cdm's and cm's)

IMPORTING NECESSARY LIBRARIES



Pandas (imported as **pd**):

Pandas is a powerful data manipulation and analysis library for Python. It provides data structures such as DataFrame and Series, which allow for easy handling and manipulation of structured data. Pandas is commonly used for tasks such as data cleaning, transformation, exploration, and analysis.

Matplotlib (imported as **plt**):

Matplotlib is a popular plotting library for Python. It provides a wide variety of functions and methods for creating static, interactive, and animated visualizations. Matplotlib can be used to generate plots, histograms, scatter plots, bar charts, line charts, and more, making it a versatile tool for data visualization.

Seaborn (imported as **sns**):

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations such as heatmaps, violin plots, pair plots, and regression plots, with options for styling and customization.

Plotly Express (imported as **px**)

Plotly Express is a high-level interface for creating interactive visualizations using Plotly, a leading visualization library for Python. Plotly Express simplifies the creation of interactive plots such as scatter plots, line plots, bar charts, and 3D plots, with built-in support for interactivity, animation, and customization.

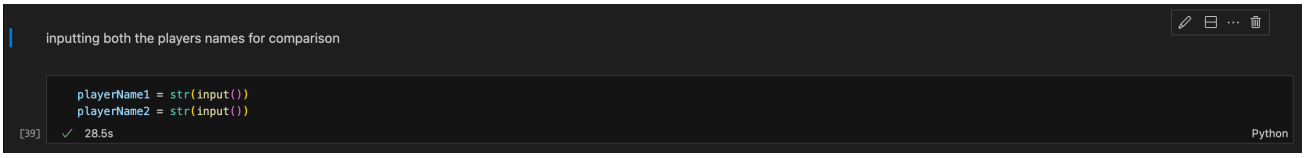
NumPy (imported as **np**)

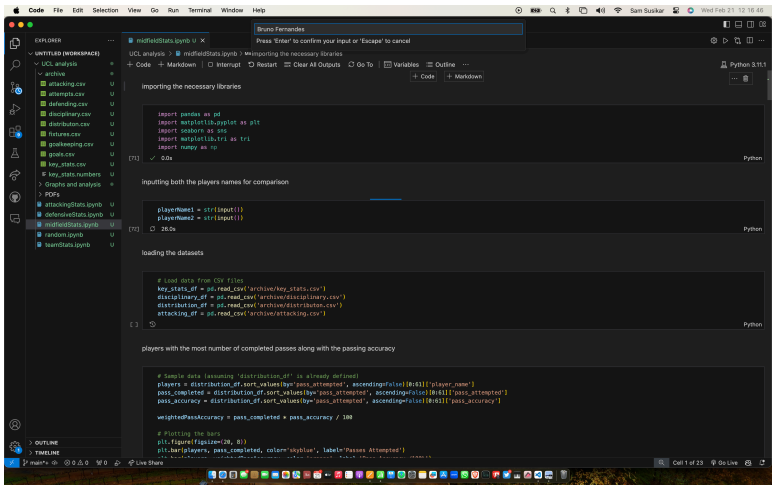
NumPy is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays, mathematical functions, linear algebra operations, random number generation, and more. NumPy is widely used for numerical analysis, scientific computing, and data manipulation tasks.

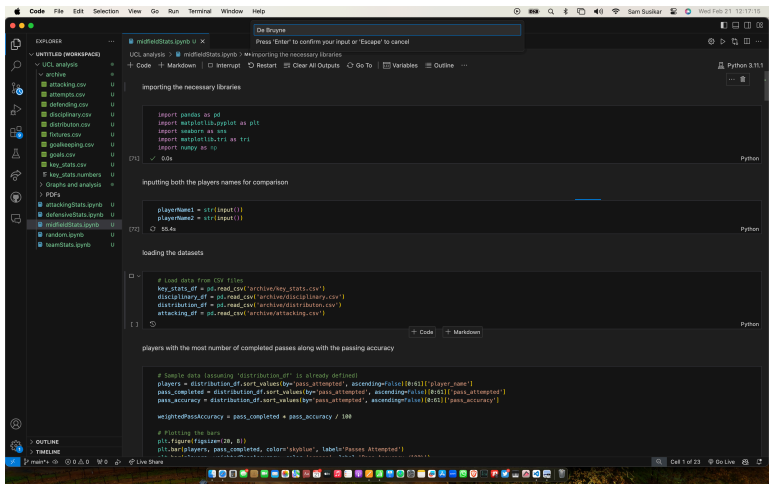
Matplotlib Triangular Mesh (imported as **tri**)**:

Matplotlib Triangular Mesh is a module within Matplotlib that provides functionality for creating and plotting triangular meshes. It allows for the creation of plots and visualizations based on triangular mesh data structures, which are commonly used in fields such as finite element analysis, computational fluid dynamics, and geospatial analysis.

TAKING IN USER INPUT FOR BOTH THE PLAYERS NAMES







The two players

LOADING THE DATASETS

loading the datasets

```
# Load data from CSV files
key_stats_df = pd.read_csv('archive/key_stats.csv')
disciplinary_df = pd.read_csv('archive/disciplinary.csv')
distribution_df = pd.read_csv('archive/distributon.csv')
attacking_df = pd.read_csv('archive/attacking.csv')
```

[40] ✓ 0.1s Python

read_csv(): This is a function provided by the Pandas library for reading data from CSV files. It's called using the dot notation (**pd.read_csv()**), where **pd** is the alias for the Pandas library.

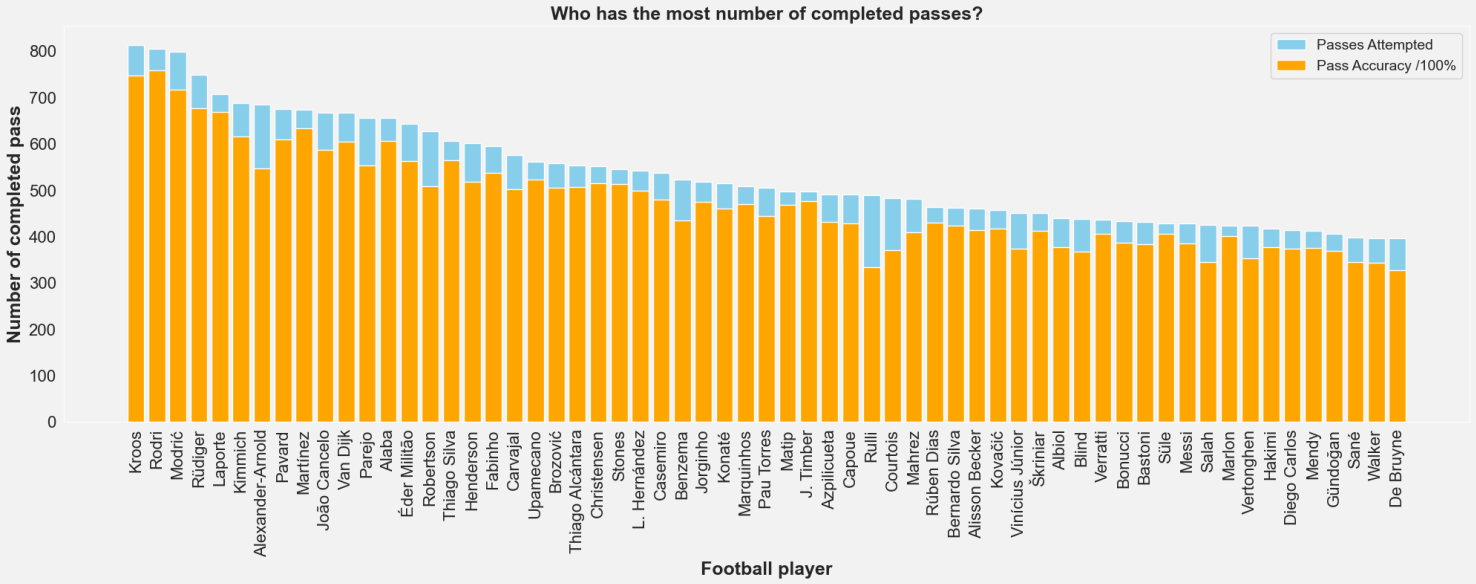
BAR PLOT TO IDENTIFY THE PLAYERS WITH THE MOST COMPLETED PASSES ALONG WITH THEIR PASSING ACCURACY

players with the most number of completed passes along with the passing accuracy

```
# Sample data (assuming 'distribution_df' is already defined)
players = distribution_df.sort_values(by='pass_attempted', ascending=False)[0:61]['player_name']
pass_completed = distribution_df.sort_values(by='pass_attempted', ascending=False)[0:61]['pass_attempted']
pass_accuracy = distribution_df.sort_values(by='pass_attempted', ascending=False)[0:61]['pass_accuracy']

weightedPassAccuracy = pass_completed * pass_accuracy / 100

# Plotting the bars
plt.figure(figsize=(20, 8))
plt.bar(players, pass_completed, color='skyblue', label='Passes Attempted')
plt.bar(players, weightedPassAccuracy, color='orange', label='Pass Accuracy /100%')

# Adding title and labels
plt.title('Who has the most number of completed passes?', fontsize=18, fontweight='bold')
plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Number of completed pass', fontsize=18, fontweight='bold')

# Adjusting tick labels
plt.xticks(rotation=90, fontsize=16)
plt.yticks(fontsize=16)

# Add legend
plt.legend(fontsize=14)

# Remove grid
plt.grid(False)

# Display the plot
plt.tight_layout()
plt.show()
```

[41] ✓ 1.3s Python

OUTPUT



Players with the most number of passes, the shaded portion in blue also represents the 100% out of which the pass accuracy is shaded in orange

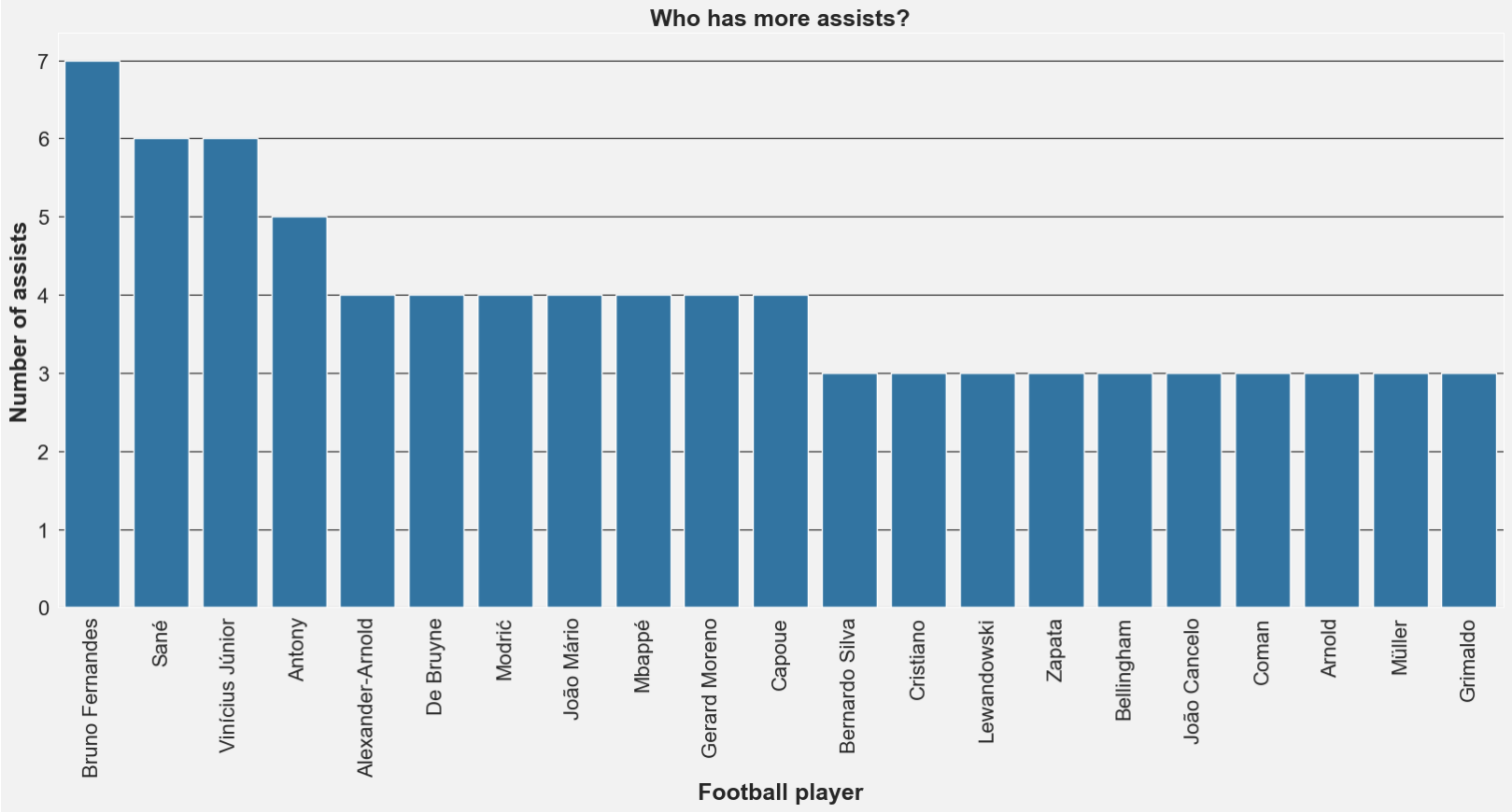
BAR PLOT TO IDENTIFY THE PLAYERS WITH THE MOST ASSISTS

most assists

```python
# Set the figure size and title
plt.figure(figsize=(20, 8))
plt.title('Who has more assists?', fontsize=18, fontweight='bold')

# Set the style of the plot
sns.set_style('darkgrid', {
    'axes.facecolor': '0.95',  # Background color of the axes
    'grid.color': '0.1',       # Color of the grid lines
    'figure.facecolor': '0.95'  # Background color of the figure
})

# Create a bar plot
sns.barplot(x=attacking_df.sort_values('assists', ascending=False)[0:21]['player_name'],  # X-axis: Player names
            y=attacking_df.sort_values('assists', ascending=False)[0:21]['assists'])       # Y-axis: Number of assists

# Set labels and ticks for x and y axes
plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Number of assists', fontsize=18, fontweight='bold')
plt.xticks(fontsize=16, rotation=90)  # Rotate x-axis labels for better readability
plt.yticks(fontsize=16)

# Show the plot
plt.show()
```

[53] ✓ 0.5s Python

OUTPUT



Bar plot for the player with the most assists

SCATTER PLOT FOR THE MINUTES PLAYED VS THE ASSISTS

scatter plot for assists vs minutes played

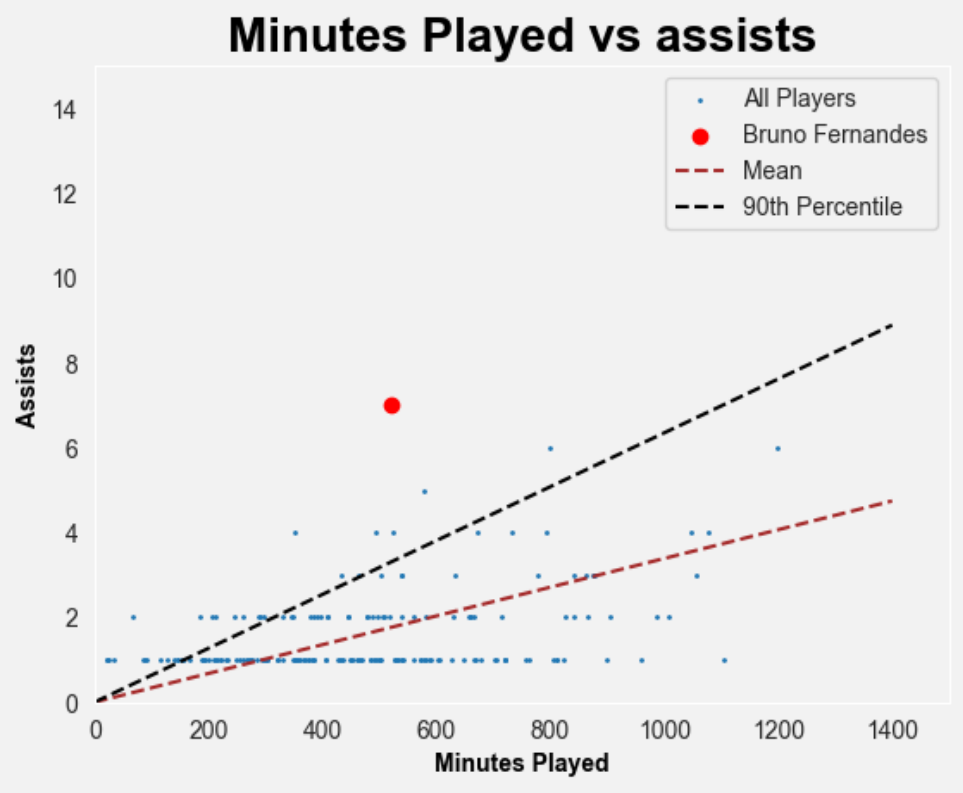
```python
# Read the CSV files
attacking_df = pd.read_csv('archive/attacking.csv')
key_stats_df = pd.read_csv('archive/key_stats.csv')

# Merge the two dataframes on the 'player_name' column
merged_df = pd.merge(attacking_df[['player_name', 'assists']], key_stats_df[['player_name', 'minutes_played']], on='player_name')
```

[59] ✓ 0.0s Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Set the limits for the plot
plt.ylim(0, 15)
plt.xlim(0, 1500)

# Set the labels for the axes (switched)
plt.xlabel("Minutes Played", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Assists", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})

# Scatter plot for all players (switched)
plt.scatter((merged_df['minutes_played']), (merged_df['assists']), label='All Players', s=1)

# Scatter plot for the specified player (switched)
player_row = merged_df[merged_df['player_name'] == playerName1]
plt.scatter((player_row['minutes_played'].iloc[0]), (player_row['assists'].iloc[0]), label=playerName1, color='red')

# Mean line (switched)
average_minutes = np.mean(merged_df['minutes_played'])
average_assists = np.mean(merged_df['assists'])
slope = average_assists / average_minutes
x_values = np.linspace(0, 1400, 50)
y_values = slope * x_values
plt.plot(x_values, y_values, color='brown', linestyle='--', label='Mean')

# 90th percentile line (switched)
percentile90_minutes = np.percentile(merged_df['minutes_played'], 90)
percentile90_assists = np.percentile(merged_df['assists'], 90)
slope90 = percentile90_assists / average_minutes
x_values90 = np.linspace(0, 1400, 50)
y_values90 = slope90 * x_values90
plt.plot(x_values90, y_values90, color='black', linestyle='--', label='90th Percentile')

# Display legend and grid
plt.legend()
plt.grid(False)
plt.title("Minutes Played vs assists", fontdict={'fontname': 'Arial', 'fontsize': 20, 'fontweight': 'bold', 'color': 'black'})

# Show the plot
plt.show()
```

[70] ✓ 0.4s Python

OUTPUT



Scatter plot representing the minutes played vs assists claimed

EXTRACTING THE TOP 90TH PERCENTILE

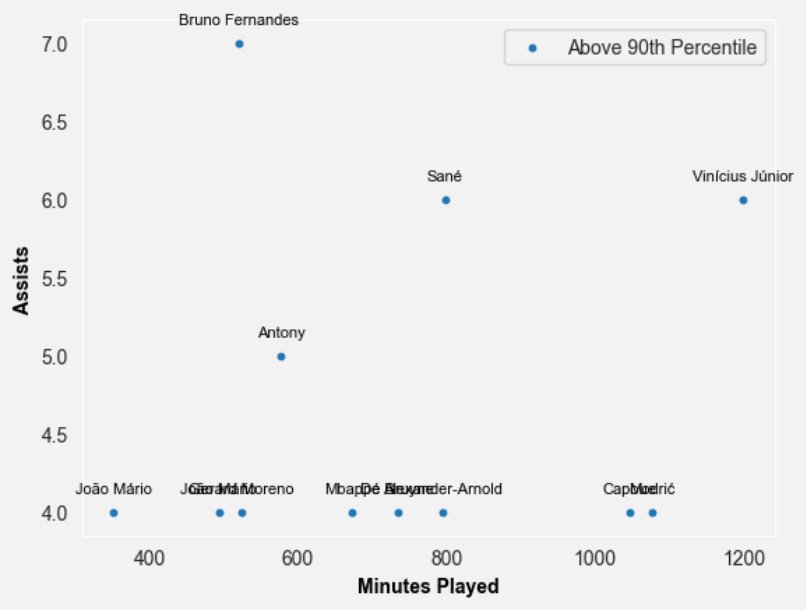
extracting the top 90 percentile

+ Code + Markdown

```
# Calculate the 90th percentile for assists
percentile90_assists = np.percentile(merged_df['assists'], 90)

# Filter players above the 90th percentile
above_90_percentile = merged_df[merged_df['assists'] > percentile90_assists]

# Plot the scatter plot for players above the 90th percentile line
plt.scatter((above_90_percentile['minutes_played']), (above_90_percentile['assists']), label='Above 90th Percentile', s=10)

# Annotate the names of players above the 90th percentile line
for index, row in above_90_percentile.iterrows():
    plt.annotate(row['player_name'], (row['minutes_played'], row['assists']), textcoords="offset points", xytext=(0,10), ha='center', fontsize=8, color='black')

# Set the labels for the axes
plt.xlabel("Minutes Played", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Assists", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})

# Display legend and grid
plt.legend()
plt.grid(False)

# Show the plot
plt.show()

# Print DataFrame with players above the 90th percentile line
print(above_90_percentile)
```

[45] ✓ 0.3s Python

OUTPUT



	player_name	assists	minutes_played
0	Bruno Fernandes	7	520
1	Vinícius Júnior	6	1199
2	Sané	6	798
3	Antony	5	577
4	Alexander-Arnold	4	794
5	De Bruyne	4	734
6	Modrić	4	1077
7	João Mário	4	493
8	João Mário	4	351
9	Mbappé	4	673
10	Gerard Moreno	4	524
11	Capoue	4	1046

stats for player 1's radar chart

STATS FOR MIDFIELDER 1'S RADAR CHART

stats for player 1's radar chart

```
# Merge the data frames using the 'player_name' column
playerGraphDataDF = pd.merge(key_stats_df[['player_name', 'minutes_played', 'distance_covered', 'match_played']],
                             disciplinary_df[['player_name', 'fouls_committed', 'fouls_suffered']],
                             on='player_name')
playerGraphDataDF = pd.merge(playerGraphDataDF,
                             distribution_df[['player_name', 'pass_accuracy']],
                             on='player_name')

# Get the row corresponding to playerName1
player_row1 = playerGraphDataDF[playerGraphDataDF['player_name'] == playerName1].iloc[0]

# Extract the matches played and minutes played stats
mp1 = player_row1['match_played']
minp1 = player_row1['minutes_played']

# Calculate ratios for the statistics to plot
playerGraphStats1 = [(player_row1['minutes_played'])/1500, (float(player_row1['distance_covered'])*90)/(minp1*15),
                     player_row1['fouls_committed']*90/minp1, player_row1['fouls_suffered']*90/(minp1*2),
                     player_row1['pass_accuracy']/100]
playerGraphStats1
```

[46] ✓ 0.0s Python

```
[0.3466666666666667,
 0.6738461538461539,
 0.8653846153846154,
 0.7788461538461539,
 0.782]
```

STATS FOR MIDFIELDER 2'S RADAR CHART

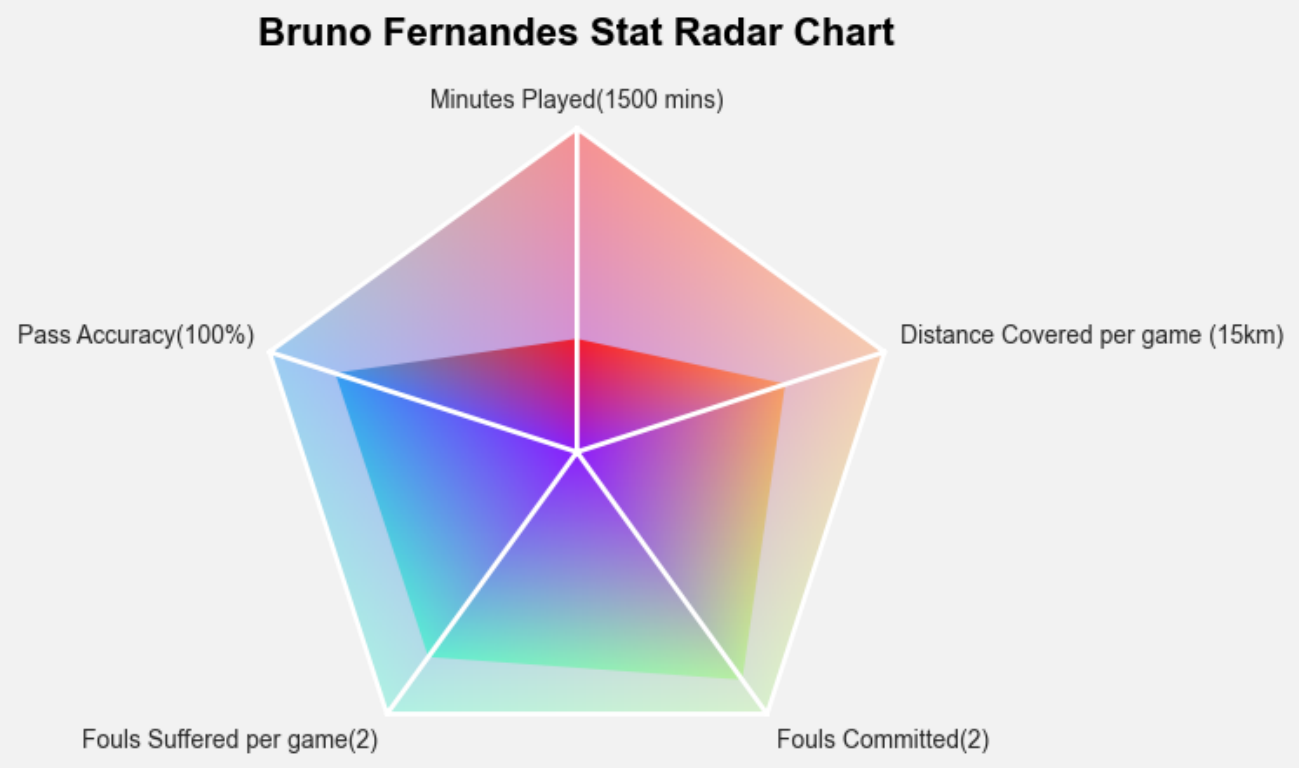
stats for player 2's radar chart

```
# Get the row corresponding to playerName2
player_row2 = playerGraphDataDF[playerGraphDataDF['player_name'] == playerName2].iloc[0]

# Extract the matches played and minutes played stats
mp2 = player_row2['match_played']
minp2 = player_row2['minutes_played']

# Calculate ratios for the statistics to plot
playerGraphStats2 = [(player_row2['minutes_played'])/1500, (float(player_row2['distance_covered'])*90)/(minp2*15),
                     player_row2['fouls_committed']*90/(minp2*2), player_row2['fouls_suffered']*90/(minp2*2),
                     player_row2['pass_accuracy']/100]

playerGraphStats2
```

[50] ✓ 0.0s Python

```
[0.48933333333333334,
 0.7806539509536785,
 0.5517711171662125,
 0.5517711171662125,
 0.826]
```

RADAR CHART FOR PLAYER 1



bracketed values are the limits defined and the shaded portion represents how well the player has done

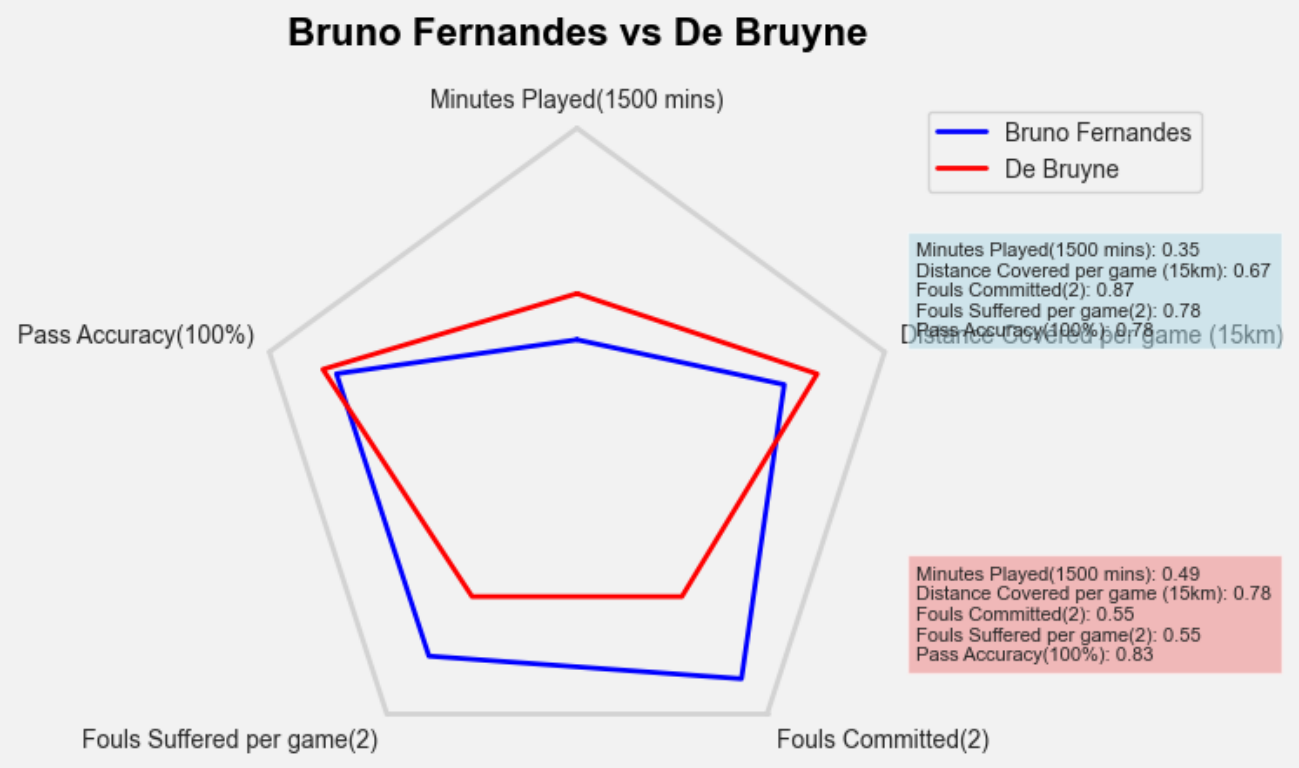
CODE FOR COMPARING PLAYER 1 AND PLAYER TWO

comparison between player 1 and player 2

```
proportions1 = playerGraphStats1
proportions2 = playerGraphStats2

# Remove a label to match the length of proportions
labels = ['Minutes Played(1500 mins)', 'Distance Covered per game (15km)',
          'Fouls Committed(2)', 'Fouls Suffered per game(2)', 'Pass Accuracy(100%)']

# Define player statistics as strings
stats1 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions1)]
stats2 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions2)]

# Join the statistics into a single string with line breaks
stats_str1 = '\n'.join(stats1)
stats_str2 = '\n'.join(stats2)

N = len(labels)
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)
x = np.sin(theta)
y = np.cos(theta)

# Append the first vertex to the end to close the polygon
x = np.append(x, x[0])
y = np.append(y, y[0])

# Plotting the filled background pentagon with 5 equilateral triangles
for i in range(N):
    plt.plot([x[i], x[(i+1)%N]], [y[i], y[(i+1)%N]], color='lightgray', lw=2)

# Plotting the first player's statistics
plt.plot(x * np.append(proportions1, proportions1[0]), y * np.append(proportions1, proportions1[0]), color='blue', lw=2, label=playerName1)

# Plotting the second player's statistics
plt.plot(x * np.append(proportions2, proportions2[0]), y * np.append(proportions2, proportions2[0]), color='red', lw=2, label=playerName2)

# Adding labels to the vertices
for label, xi, yi in zip(labels, x[:-1], y[:-1]):
    plt.text(xi * 1.05, yi * 1.05, label,
             ha='left' if xi > 0.1 else 'right' if xi < -0.1 else 'center',
             va='bottom' if yi > 0.1 else 'top' if yi < -0.1 else 'center')

# Adding legend and placing it on the extreme right
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Setting plot properties
plt.axis('off')
plt.gca().set_aspect('equal')
plt.title(playerName1 +' vs '+playerName2+'\n', fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Adding a text box with player statistics
plt.text(1.05, 0.5, stats_str1, bbox=dict(facecolor='lightblue', alpha=0.5), fontsize=8, verticalalignment='center')
plt.text(1.05, -0.5, stats_str2, bbox=dict(facecolor='lightcoral', alpha=0.5), fontsize=8, verticalalignment='center')

plt.show()
```

[65] ✓ 0.3s Python

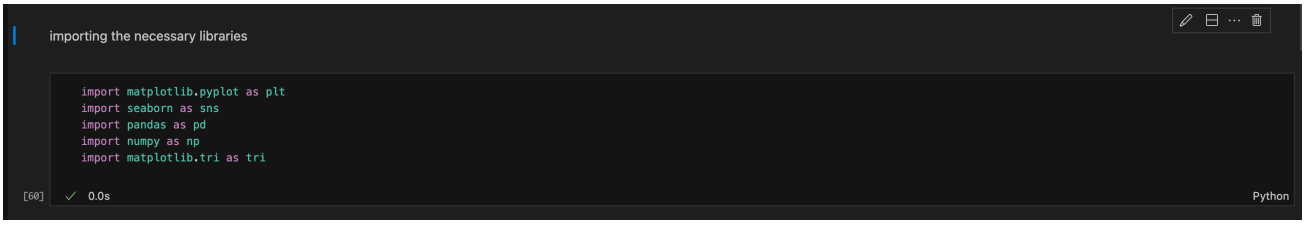
RADAR CHART FOR PLAYER 1 VS PLAYER 2



DEFENSIVESTATS.IPYNB

the third notebook is the defensiveStats notebook and I have used it in order to analyse the defemnsive output of the defenders and defensive midfielders in the league. These include the all the different defenders (CB's, RB's, LB's, RWB's, LWB's) and defensive midfielders (CDM's)

IMPORTING THE NECESSARY LIBRARIES



Pandas (imported as **pd**):

Pandas is a powerful data manipulation and analysis library for Python. It provides data structures such as DataFrame and Series, which allow for easy handling and manipulation of structured data. Pandas is commonly used for tasks such as data cleaning, transformation, exploration, and analysis.

Matplotlib (imported as **plt**):

Matplotlib is a popular plotting library for Python. It provides a wide variety of functions and methods for creating static, interactive, and animated visualizations. Matplotlib can be used to generate plots, histograms, scatter plots, bar charts, line charts, and more, making it a versatile tool for data visualization.

Seaborn (imported as **sns**):

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations such as heatmaps, violin plots, pair plots, and regression plots, with options for styling and customization.

NumPy (imported as **np**)

NumPy is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays, mathematical functions, linear algebra operations,

random number generation, and more. NumPy is widely used for numerical analysis, scientific computing, and data manipulation tasks.

Matplotlib Triangular Mesh (imported as **tri**):

Matplotlib Triangular Mesh is a module within Matplotlib that provides functionality for creating and plotting triangular meshes. It allows for the creation of plots and visualizations based on triangular mesh data structures, which are commonly used in fields such as finite element analysis, computational fluid dynamics, and geospatial analysis.

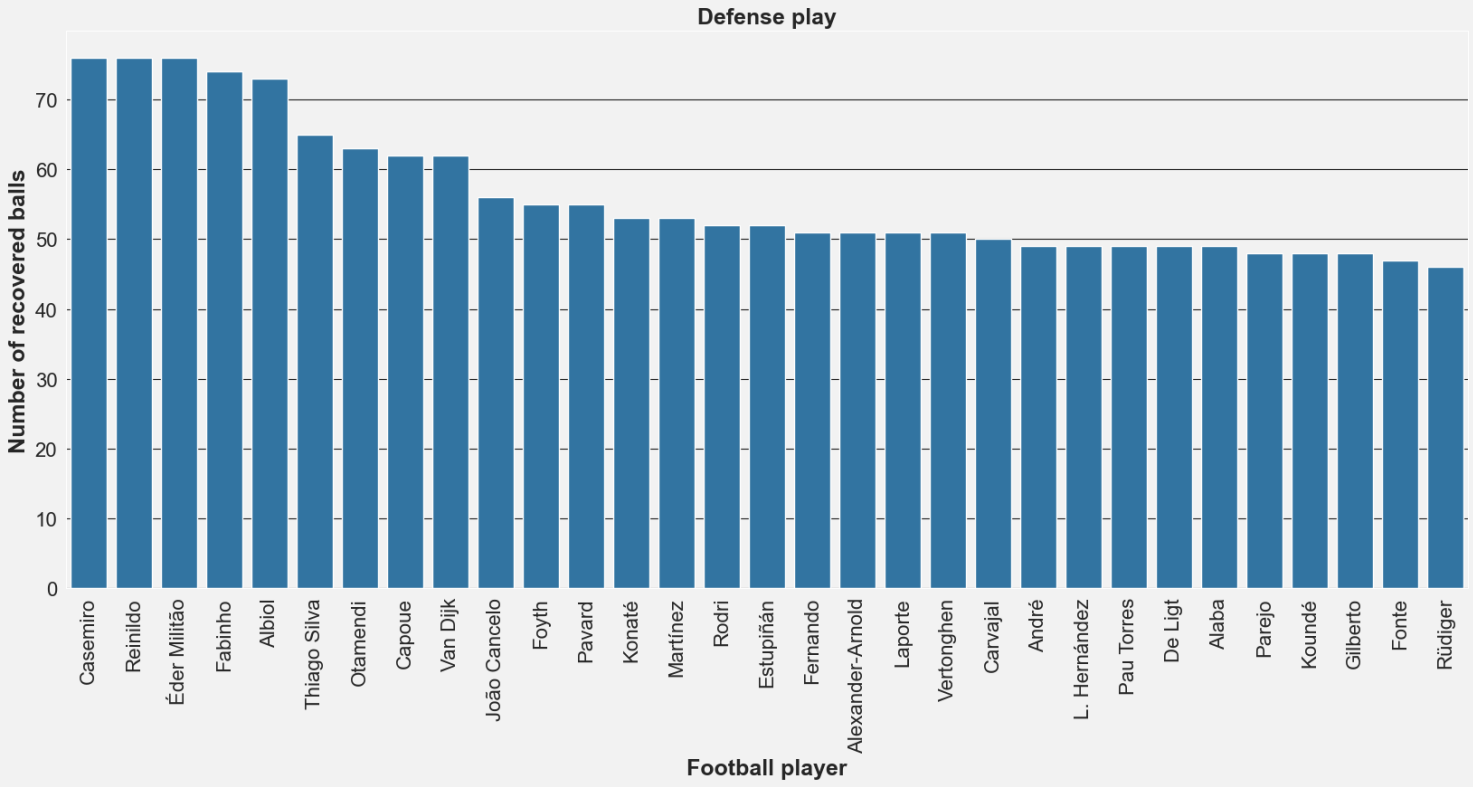
BAR CHART FOR PLAYERS WITH THE MOST BALL RECOVERIES

top defensive players with most ball recoveries

```
defendingDF = pd.read_csv('archive/defending.csv')

plt.figure(figsize=(20, 8))
plt.title('Defense play', fontsize=18, fontweight='bold')
sns.set_style('darkgrid',
              {'axes.facecolor': '0.95',
               'grid.color': '0.1',
               'figure.facecolor': '0.95'})
sns.barplot(x=defendingDF.sort_values('balls_recoverd', ascending=False)[0:31]['player_name'],
            y=defendingDF.sort_values('balls_recoverd', ascending=False)[0:31]['balls_recoverd'])
plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Number of recovered balls', fontsize=18, fontweight='bold')
plt.xticks(fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.show()
```

[61] ✓ 0.5s Python

OUTPUT



Players with the most balls recovered

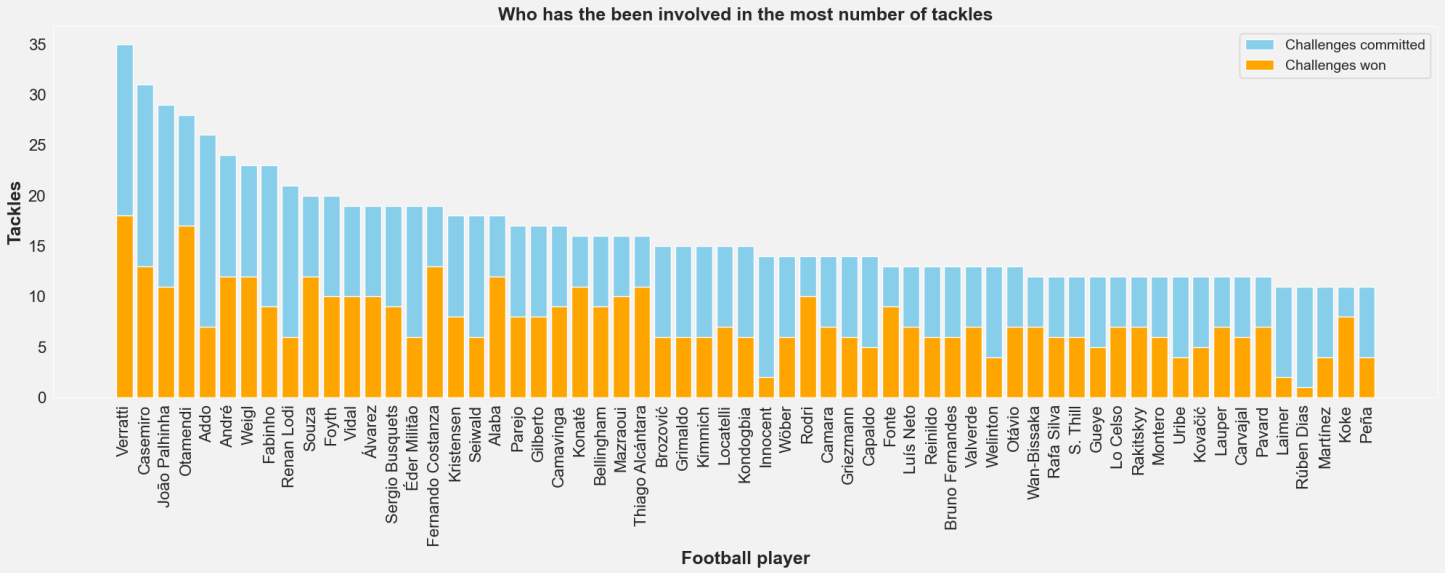
TACKLES VS TACKLES WON BAR PLOT

most tackles vs tackles won bar plot

```python
# Sample data (assuming 'distribution_df' is already defined)
players = defendingDF.sort_values(by='tackles', ascending=False)[0:61]['player_name']
tackles = defendingDF.sort_values(by='tackles', ascending=False)[0:61]['tackles']
tacklesLost = defendingDF.sort_values(by='tackles', ascending=False)[0:61]['t_won']



# Plotting the bars
plt.figure(figsize=(20, 8))
plt.bar(players, tackles, color='skyblue', label='Challenges committed')
plt.bar(players, tacklesLost, color='orange', label='Challenges won')

# Adding title and labels
plt.title('Who has the been involved in the most number of tackles', fontsize=18, fontweight='bold')
plt.xlabel('Football player', fontsize=18, fontweight='bold')
plt.ylabel('Tackles', fontsize=18, fontweight='bold')

# Adjusting tick labels
plt.xticks(rotation=90, fontsize=16)
plt.yticks(fontsize=16)

# Add legend
plt.legend(fontsize=14)

# Remove grid
plt.grid(False)

# Display the plot
plt.tight_layout()
plt.show()
```

[62] ✓ 0.9s Python

OUTPUT

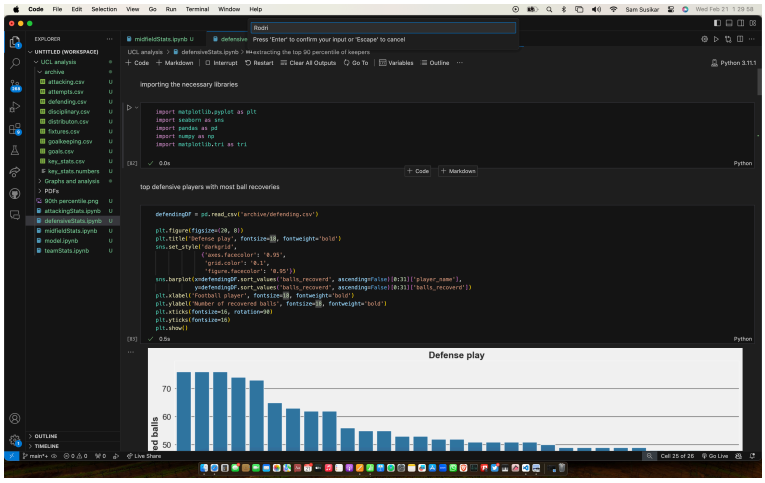


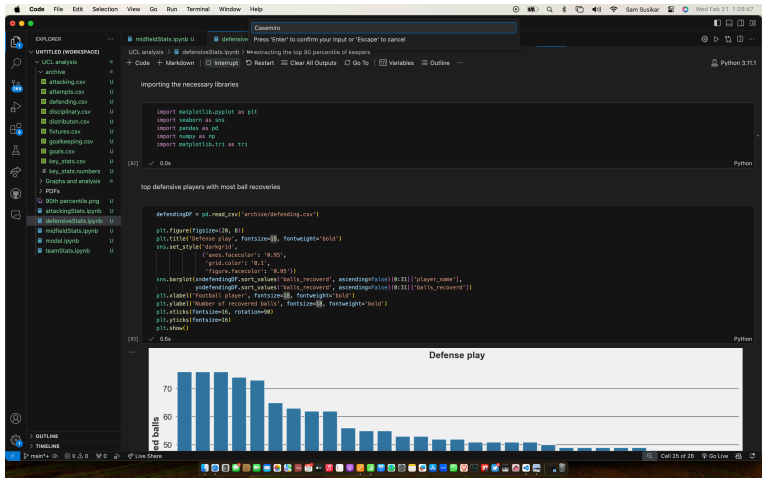
TAKING TWO PLAYERS NAMES AS INPUT

inputting two players

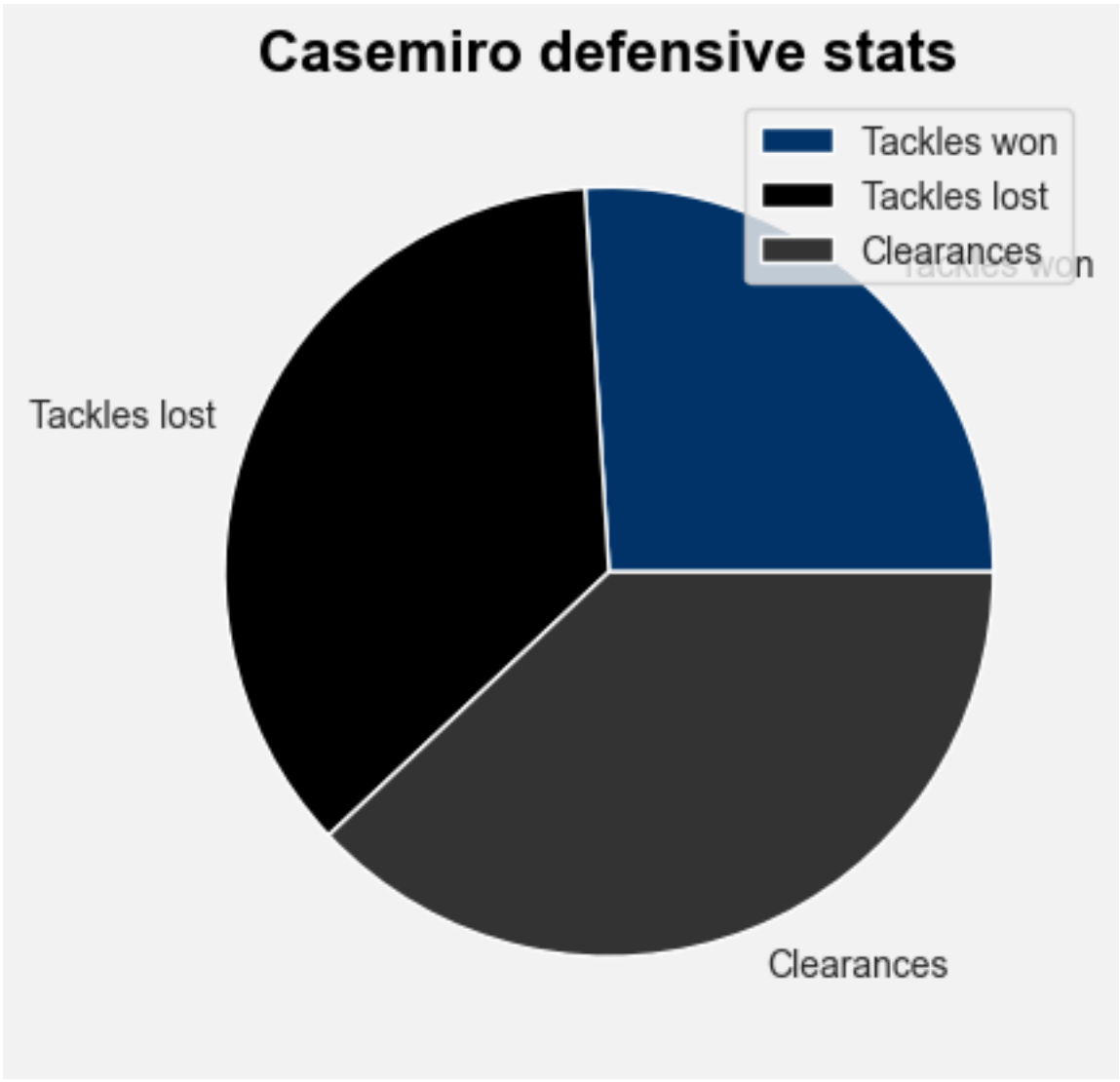
```python
playerName = str(input())
playerName2 = str(input())
```

[63] ✓ 5.1s Python





PIE CHART FOR TACKLES WON VS TACKLES LOST VS CLEARANCES



SCATTER PLOT FOR TACKLES ATTEMPTED VS TACKLES WON

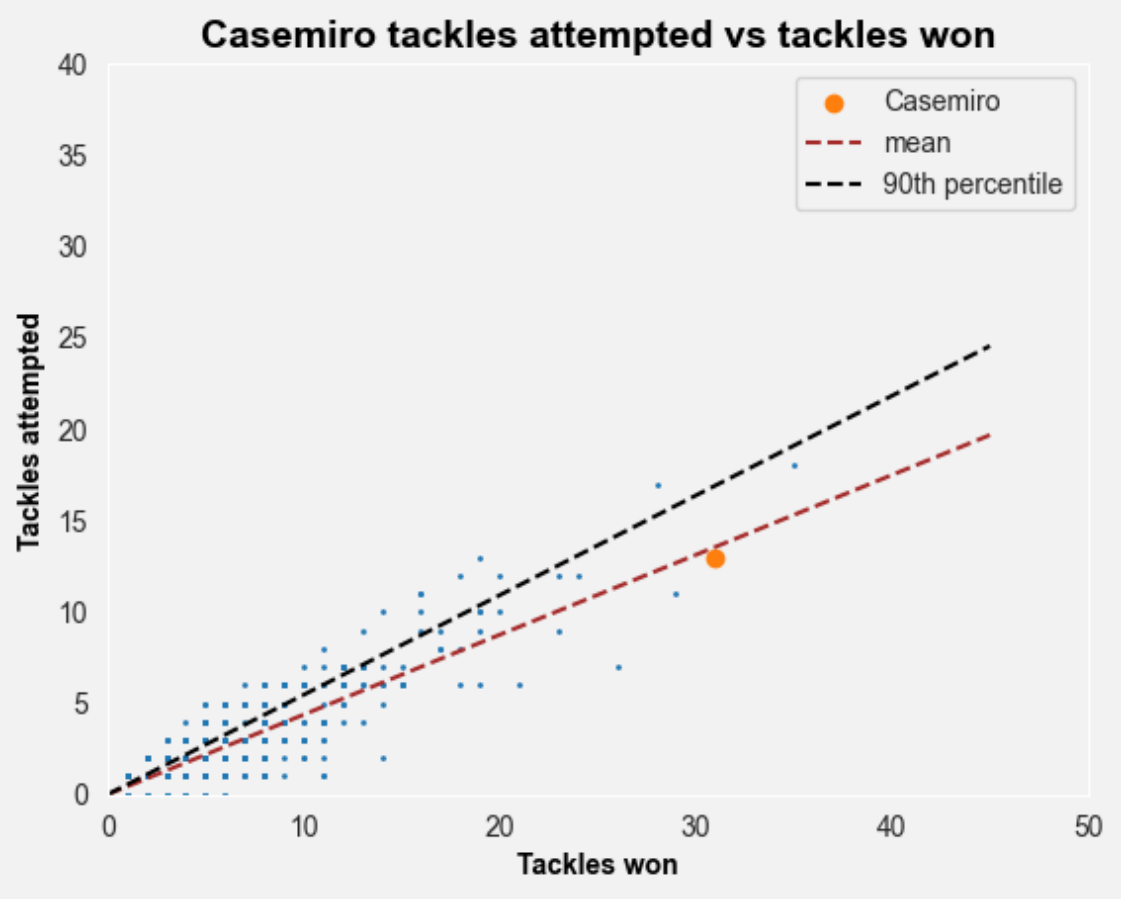
tackles attempted vs tackles won with mean and 90th percentiles line

+ Code + Markdown

```
# x and y limits based on the given dataset
plt.ylim(0,40)
plt.xlim(0,50)


# x and y labels
plt.xlabel("Tackles won", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Tackles attempted", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})

# merging minutes played and attempts from two different datasets

# scatter plot 1 for every player in the league
plt.scatter(defendingDF['tackles'], defendingDF['t_won'],s=1)
tackles = defendingDF['tackles'].iloc[0]
tacklesWon = defendingDF['t_won'].iloc[0]

# scatter plot 2 for given searched player
plt.scatter(player_row1['tackles'], player_row1['t_won'], label=playerName)
plt.title(playerName+" tackles attempted vs tackles won", fontdict={'fontname': 'Arial', 'fontsize': 14, 'fontweight': 'bold', 'color': 'black'})



# Mean line
average_y = np.mean(defendingDF['t_won'])
average_x = np.mean(defendingDF['tackles'])
slope=(average_y)/(average_x)
x_values = np.linspace(0, 45, 50)
y_values = slope*(x_values)
plt.plot(x_values, y_values, color='brown',linestyle='--',label='mean')


# 90th percentile line
percentile90_y =np.percentile(defendingDF['t_won'], 90)
percentile90_x =np.percentile(defendingDF['tackles'], 90)
slope90 = percentile90_y/percentile90_x
x_values90 = np.linspace(0, 45, 50)
y_values90 = slope90*(x_values90)
plt.plot(x_values90, y_values90, color='black',linestyle='--',label='90th percentile')

plt.legend()

plt.grid(False)
```

[65] ✓ 0.3s Python

OUTPUT



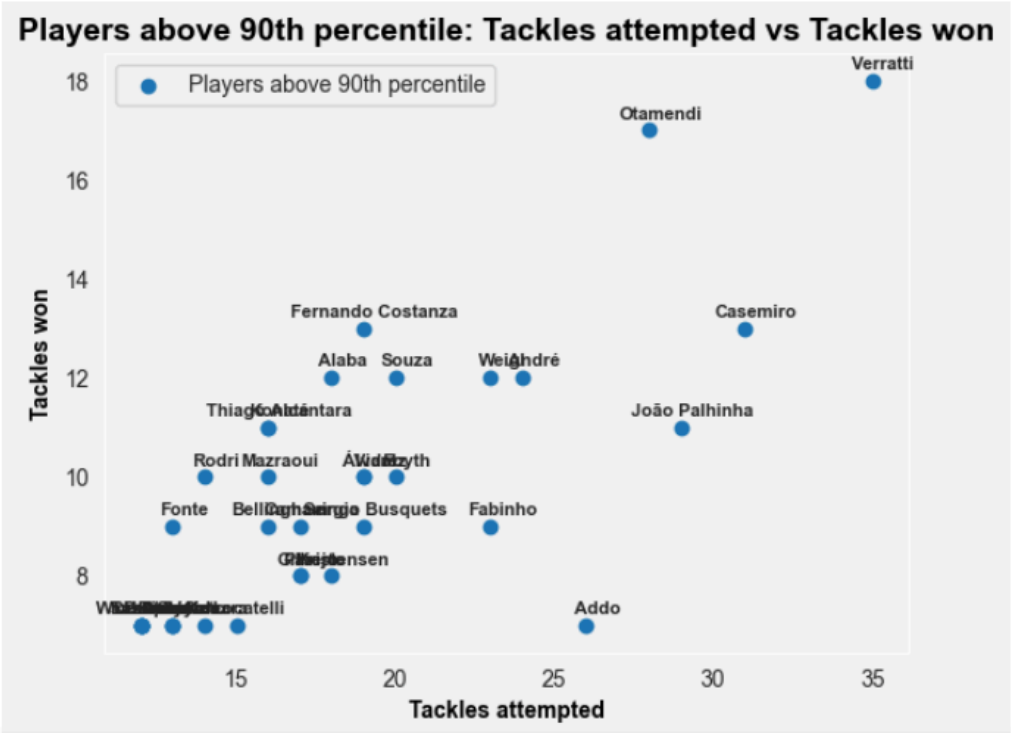
EXTRACTING THE TOP 90 PERCENT

seperating the players above the 90th percentile

```
# Filter players above the 90th percentile line
players_above_90th = defendingDF[(defendingDF['tackles'] > percentile90_x) & (defendingDF['t_won'] > percentile90_y)]

# Display their information in a DataFrame
players_df = players_above_90th[['player_name', 'tackles', 't_won']]
print(players_df)

# Plot their stats separately and annotate their names
plt.figure()
plt.scatter(players_above_90th['tackles'], players_above_90th['t_won'], label='Players above 90th percentile')
plt.xlabel("Tackles attempted", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Tackles won", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.title("Players above 90th percentile: Tackles attempted vs Tackles won", fontdict={'fontname': 'Arial', 'fontsize': 14, 'fontweight': 'bold', 'color': 'black'})
plt.legend()

# Annotate player names
for index, player in players_above_90th.iterrows():
    plt.annotate(player['player_name'], (player['tackles'], player['t_won']), textcoords="offset points", xytext=(5,5), ha='center', fontsize=8, fontweight='bold')

plt.grid(False)

plt.show()
```

[66] ✓ 0.3s Python

OUTPUT



	player_name	tackles	t_won
0	Casemiro	31	13
3	Fabinho	23	9
6	Otamendi	28	17
10	Foyth	20	10
11	Pavard	12	7
12	Konaté	16	11
14	Rodri	14	10
21	André	24	12
22	Alaba	18	12
26	Gilberto	17	8
27	Parejo	17	8
29	Fonte	13	9
36	Lauper	12	7
41	Camara	14	7
48	Souza	20	12
49	Thiago Alcántara	16	11
50	Mazraoui	16	10
53	Fernando Costanza	19	13
69	João Palhinha	29	11
70	Kristensen	18	8
75	Weigl	23	12
94	Rakitskyy	12	7
109	Vidal	19	10
120	Álvarez	19	10
...			
178	Verratti	35	18
188	Lo Celso	12	7
204	Camavinga	17	9
205	Otávio	13	7

RADAR CHART STATS FOR PLAYER 1

stats for player 1

```
import pandas as pd

# Read key stats data from CSV file
key_stats_df = pd.read_csv('archive/key_stats.csv')

# Merge key stats data with defending data based on player name
playerGraphDataDF = pd.merge(key_stats_df[['player_name', 'distance_covered', 'match_played']],
                             defendingDF[['player_name', 'balls_recovered', 'tackles', 't_won', 'clearance_attempted']],
                             on='player_name')

# Replace '-' with 0.0 in the distance_covered column
playerGraphDataDF['distance_covered'] = playerGraphDataDF['distance_covered'].replace('-', 0.0)

# Extract row for the specified player
player_row1 = playerGraphDataDF[playerGraphDataDF['player_name'] == playerName].iloc[0]

# Convert distance_covered column to float after removing commas
playerGraphDataDF['distance_covered'] = playerGraphDataDF['distance_covered'].str.replace(',', '').astype(float)

# Calculate the proportion of each stat per match played
mp = int(player_row1['match_played'])
playerGraphStats = [float(player_row1['distance_covered']) / 15,
                    float(player_row1['balls_recovered']) / 10,
                    player_row1['tackles'] / 4,
                    player_row1['t_won'] / 3,
                    player_row1['clearance_attempted'] / 3]

# Normalize the stats based on matches played
playerGraphStats = [x / mp for x in playerGraphStats]

# Display the normalized stats
playerGraphStats
```

[67] ✓ 0.0s Python

```
[0.6521212121212121,
 0.6909090909090909,
 0.7045454545454546,
 0.3939393939393939,
 0.5757575757575757]
```

RADAR CHART STATS FOR PLAYER 2

stats for player 2

```
import pandas as pd

# Read key stats data from CSV file
key_stats_df = pd.read_csv('archive/key_stats.csv')

# Merge key stats data with defending data based on player name
playerGraphDataDF2 = pd.merge(key_stats_df[['player_name', 'distance_covered', 'match_played']],
                              defendingDF[['player_name', 'balls_recovered', 'tackles', 't_won', 'clearance_attempted']],
                              on='player_name')

# the datasent had hyphens (-) in the places where no data was available so we replace those values with a 0.0 floating point value by the following line of code

# Replace '-' with 0.0 in the distance_covered column
playerGraphDataDF2['distance_covered'] = playerGraphDataDF2['distance_covered'].replace('-', 0.0)

# Extract row for the specified player 2
player_row2 = playerGraphDataDF2[playerGraphDataDF2['player_name'] == playerName2].iloc[0]

# Convert distance_covered column to float after removing commas
playerGraphDataDF2['distance_covered'] = playerGraphDataDF2['distance_covered'].str.replace(',', '').astype(float)

# Calculate the proportion of each stat per match played
mp2 = int(player_row2['match_played'])
playerGraphStats2 = [float(player_row2['distance_covered']) / 15,
                     float(player_row2['balls_recovered']) / 10,
                     player_row2['tackles'] / 4,
                     player_row2['t_won'] / 3,
                     player_row2['clearance_attempted'] / 3]

# Normalize the stats based on matches played
playerGraphStats2 = [x / mp2 for x in playerGraphStats2]

# Display the normalized stats
playerGraphStats2
```

[68] ✓ 0.0s Python

```
[0.764, 0.52, 0.35, 0.33333333333333337, 0.16666666666666669]
```

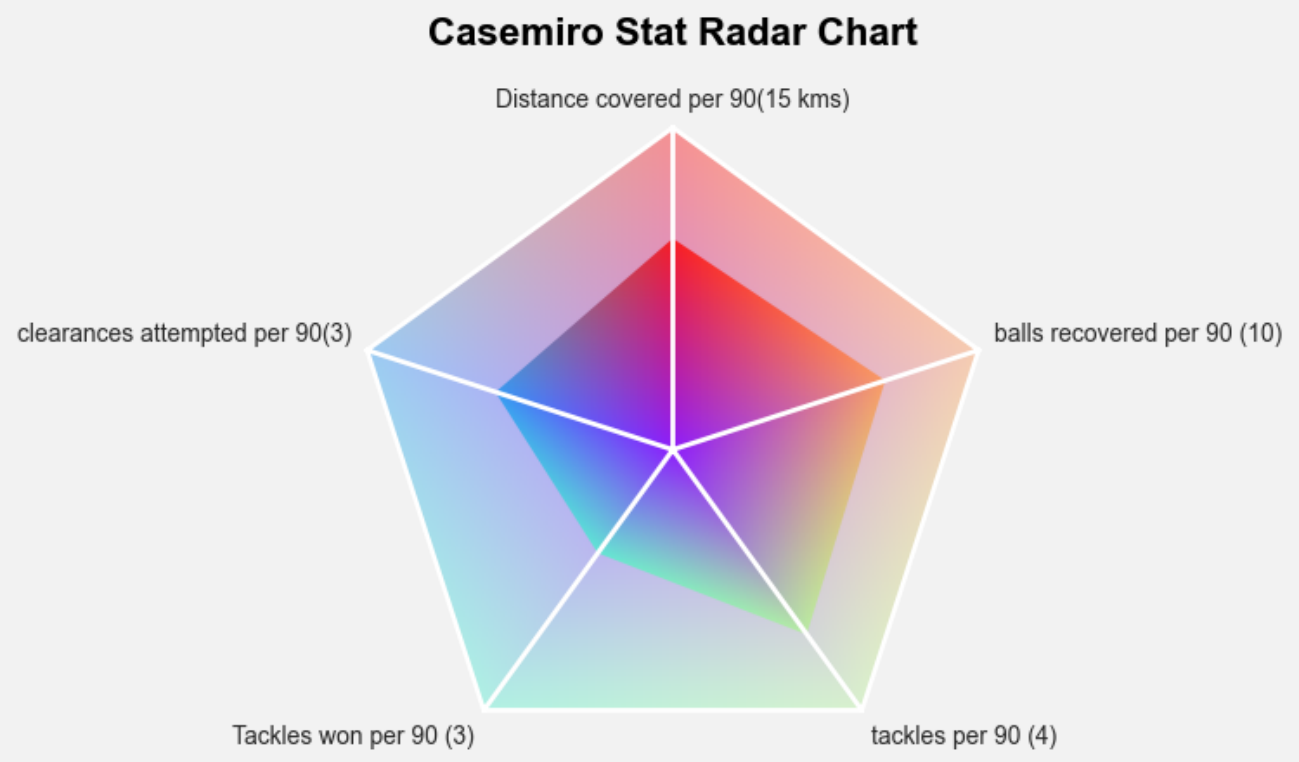
PLOTTING THE RADAR CHART FOR PLAYER 1

plotting a pentagonal radar chart for player 1

```python
# Define the proportions and labels for the radar chart
proportions = playerGraphStats  # The normalized statistics for the player
labels = ['Distance covered per 90 (15 km)', 'Balls recovered per 90 (10)', 'Tackles per 90 (4)',
          'Tackles won per 90 (3)', 'Clearances attempted per 90 (3)']

# Number of vertices in the radar chart
N = len(proportions)

# Extend the proportions array to close the radar chart
proportions = np.append(proportions, 1)

# Generate angles for each vertex of the radar chart
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)

# Calculate the x and y coordinates of the vertices
x = np.append(np.sin(theta), 0)  # x-coordinate
y = np.append(np.cos(theta), 0)  # y-coordinate

# Define the triangles to create the background and foreground of the radar chart
triangles = [[N, i, (i + 1) % N] for i in range(N)]
triang_backgr = tri.Triangulation(x, y, triangles)  # Background triangles
triang_foregr = tri.Triangulation(x * proportions, y * proportions, triangles)  # Foreground triangles

# Define the colormap and colors for the radar chart
cmap = plt.cm.rainbow_r  # Choose a colormap
colors = np.linspace(0, 1, N + 1)  # Define colors for vertices

# Plot the background and foreground triangles
plt.tripcolor(triang_backgr, colors, cmap=cmap, shading='gouraud', alpha=0.4)
plt.tripcolor(triang_foregr, colors, cmap=cmap, shading='gouraud', alpha=0.8)

# Plot the outline of the radar chart
plt.triplot(triang_backgr, color='white', lw=2)

# Add labels to each vertex of the radar chart
for label, color, xi, yi in zip(labels, colors, x, y):
    plt.text(xi * 1.05, yi * 1.05, label,
             ha='left' if xi > 0.1 else 'right' if xi < -0.1 else 'center',
             va='bottom' if yi > 0.1 else 'top' if yi < -0.1 else 'center')

# Turn off the axes and set the aspect ratio to equal
plt.axis('off')
plt.gca().set_aspect('equal')

# Add a title to the radar chart
plt.title(playerName + ' Stat Radar Chart\n', fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Show the radar chart
plt.show()
```

[73] ✓ 0.3s Python

OUTPUT



PLAYER 1 VS PLAYER 2 RADAR CHART

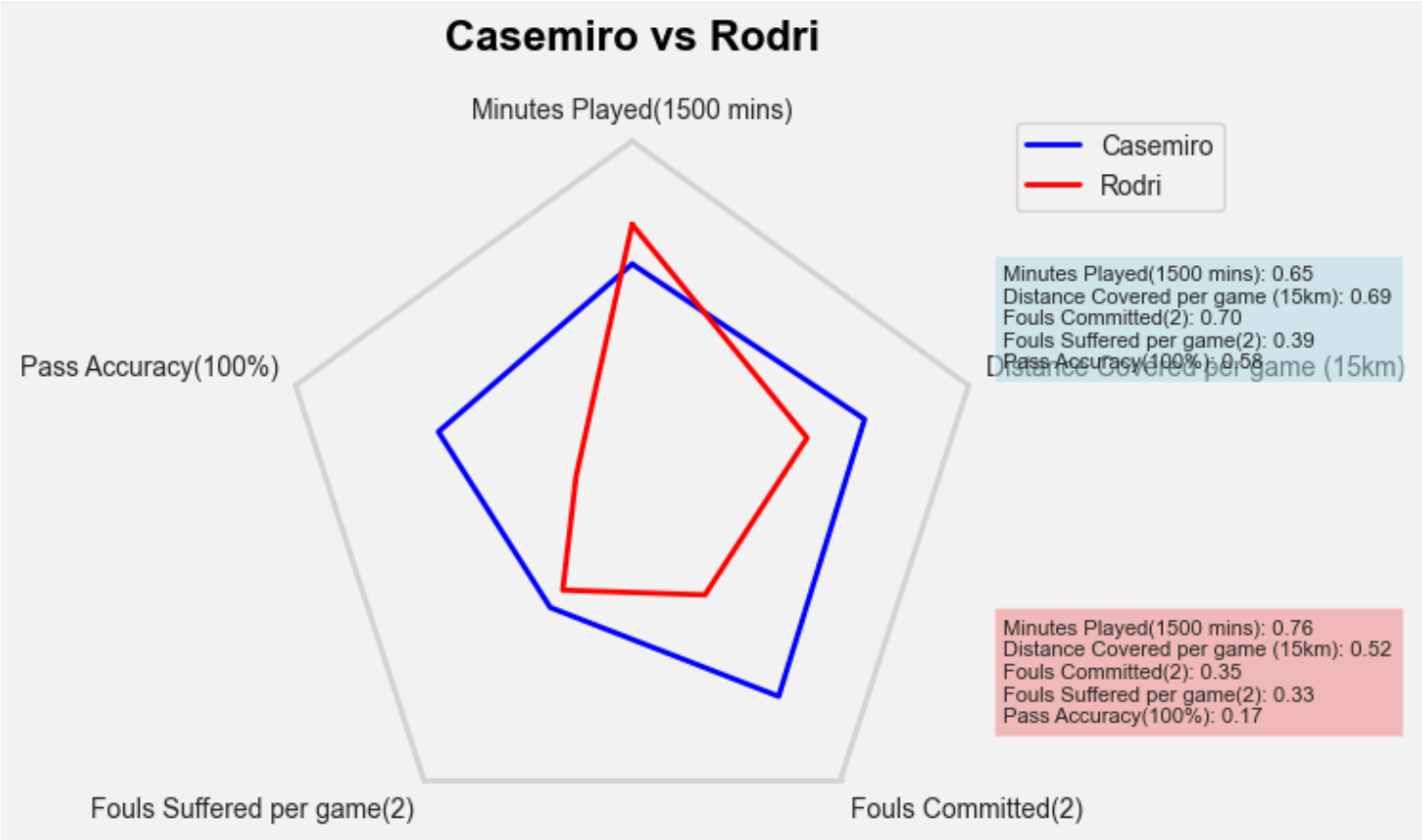
radar chart player 1 vs player 2

+ Code + Markdown

```
proportions1 = playerGraphStats
proportions2 = playerGraphStats2

labels = ['Minutes Played(1500 mins)', 'Distance Covered per game (15km)',
          'Fouls Committed(2)', 'Fouls Suffered per game(2)', 'Pass Accuracy(100%)']

# Define player statistics as strings
stats1 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions1)]
stats2 = [f'{label}: {value:.2f}' for label, value in zip(labels, proportions2)]

# Join the statistics into a single string with line breaks
stats_str1 = '\n'.join(stats1)
stats_str2 = '\n'.join(stats2)

N = len(labels)
theta = np.linspace(0, 2 * np.pi, N, endpoint=False)
x = np.sin(theta)
y = np.cos(theta)

# Append the first vertex to the end to close the polygon
x = np.append(x, x[0])
y = np.append(y, y[0])

# Plotting the filled background pentagon with 5 equilateral triangles
for i in range(N):
    plt.plot([x[i], x[(i+1)%N]], [y[i], y[(i+1)%N]], color='lightgray', lw=2)

# Plotting the first player's statistics
plt.plot(x * np.append(proportions1, proportions1[0]), y * np.append(proportions1, proportions1[0]), color='blue', lw=2, label=playerName)

# Plotting the second player's statistics
plt.plot(x * np.append(proportions2, proportions2[0]), y * np.append(proportions2, proportions2[0]), color='red', lw=2, label=playerName2)

# Adding labels to the vertices
for label, xi, yi in zip(labels, x[:-1], y[:-1]):
    plt.text(xi * 1.05, yi * 1.05, label,
             ha='left' if xi > 0.1 else 'right' if xi < -0.1 else 'center',
             va='bottom' if yi > 0.1 else 'top' if yi < -0.1 else 'center')

# Adding legend and placing it on the extreme right
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Setting plot properties
plt.axis('off')
plt.gca().set_aspect('equal')
plt.title(playerName +' vs '+playerName2+'\n', fontdict={'fontname': 'Arial', 'fontsize': 16, 'fontweight': 'bold', 'color': 'black'})

# Adding a text box with player statistics
plt.text(1.05, 0.5, stats_str1, bbox=dict(facecolor='lightblue', alpha=0.5), fontsize=8, verticalalignment='center')
plt.text(1.05, -0.5, stats_str2, bbox=dict(facecolor='lightcoral', alpha=0.5), fontsize=8, verticalalignment='center')

plt.show()
```

[70] ✓ 0.3s Python

OUTPUT

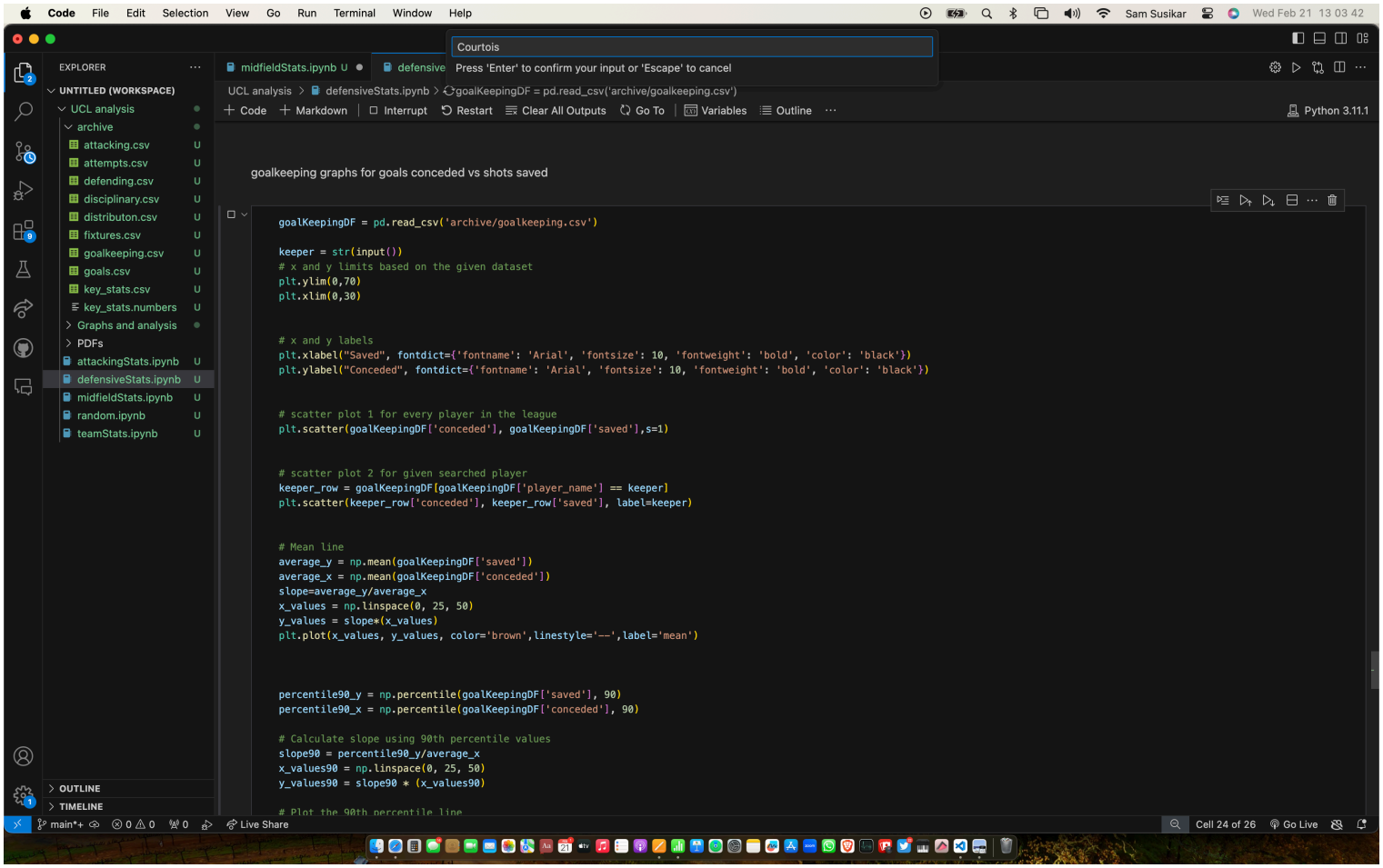


GOALKEEPING STATS: GOALS CONCEDED VS SHOTS SAVED

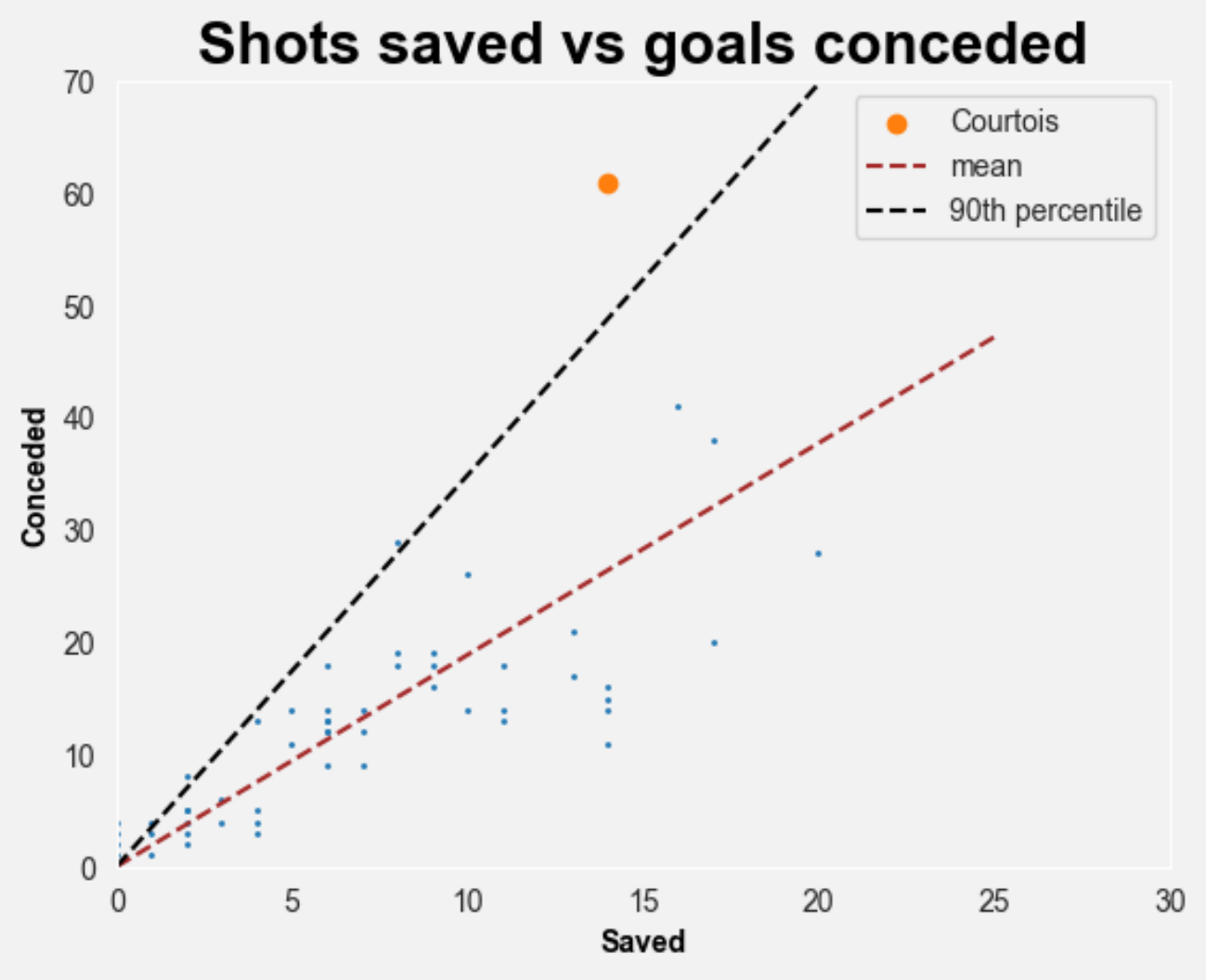
goalkeeping graphs for goals conceded vs shots saved

```
goalKeepingDF = pd.read_csv('archive/goalkeeping.csv')

keeper = str(input())
# x and y limits based on the given dataset
plt.ylim(0,70)
plt.xlim(0,30)


# x and y labels
plt.xlabel("Saved", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})
plt.ylabel("Conceded", fontdict={'fontname': 'Arial', 'fontsize': 10, 'fontweight': 'bold', 'color': 'black'})


# scatter plot 1 for every player in the league
plt.scatter(goalKeepingDF['conceded'], goalKeepingDF['saved'],s=1)


# scatter plot 2 for given searched player
keeper_row = goalKeepingDF[goalKeepingDF['player_name'] == keeper]
plt.scatter(keeper_row['conceded'], keeper_row['saved'], label=keeper)


# Mean line
average_y = np.mean(goalKeepingDF['saved'])
average_x = np.mean(goalKeepingDF['conceded'])
slope=average_y/average_x
x_values = np.linspace(0, 25, 50)
y_values = slope*(x_values)
plt.plot(x_values, y_values, color='brown',linestyle='--',label='mean')



percentile90_y = np.percentile(goalKeepingDF['saved'], 90)
percentile90_x = np.percentile(goalKeepingDF['conceded'], 90)

# Calculate slope using 90th percentile values
slope90 = percentile90_y/average_x
x_values90 = np.linspace(0, 25, 50)
y_values90 = slope90 * (x_values90)

# Plot the 90th percentile line
plt.plot(x_values90, y_values90, color='black', linestyle='--', label='90th percentile')
plt.legend()
plt.title('Shots saved vs goals conceded', fontdict={'fontname': 'Arial', 'fontsize': 20, 'fontweight': 'bold', 'color': 'black'})

plt.grid(False)
```

[71] ✓ 3.0s Python



OUTPUT


Shots saved vs goals conceded
Courtois
mean
90th percentile
Conceded
Saved
0
10
20
30
40
50
60
70
0
5
10
15
20
25
30

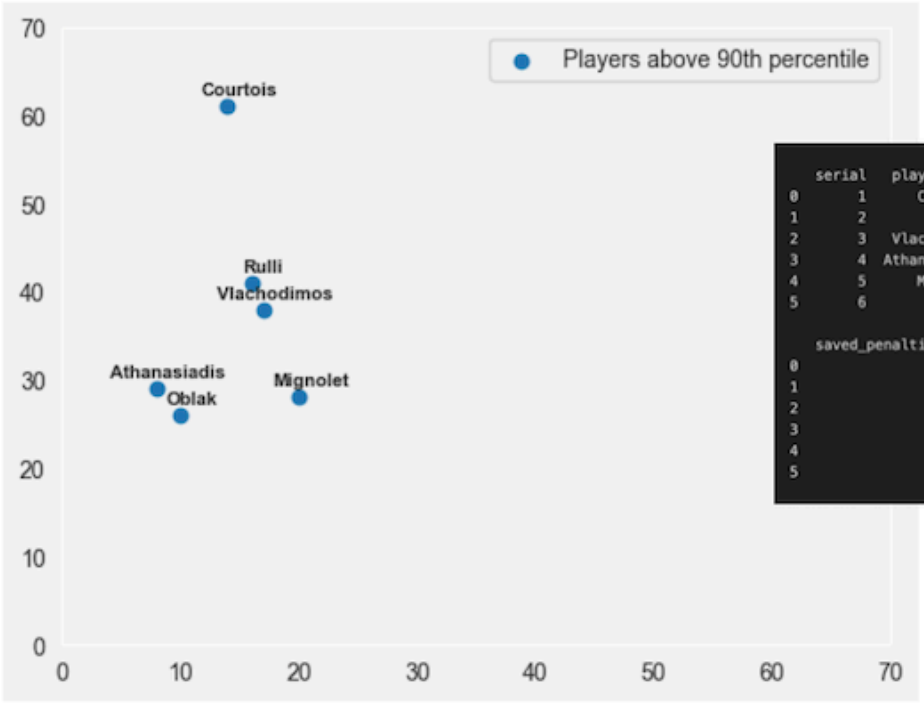
EXTRACTING THE TOP 90 PERCENT

extracting the top 90 percentile of keepers

```python
# Calculate 90th percentile values
percentile90_saved = np.percentile(goalKeepingDF['saved'], 90)
percentile90_conceded = np.percentile(goalKeepingDF['conceded'], 90)

# Filter players above the 90th percentile line
players_above_90th = goalKeepingDF[(goalKeepingDF['saved'] > percentile90_saved)]


plt.ylim(0,70)
plt.xlim(0,70)

# Plot players above the 90th percentile line
plt.scatter(players_above_90th['conceded'], players_above_90th['saved'], label='Players above 90th percentile')

# Annotate player names
for index, player in players_above_90th.iterrows():
    plt.annotate(player['player_name'], (player['conceded'], player['saved']), textcoords="offset points", xytext=(5,5), ha='center', fontsize=8, fontweight='bold')

plt.legend()

plt.grid(False)
plt.show()

# Display stats of players above 90th percentile in a dataframe
print(players_above_90th)
```

[72] ✓ 0.2s Python

OUTPUT



	serial	player_name	club	position	saved	conceded	\
0	1	Courtois	Real Madrid	Goalkeeper	61	14	
1	2	Rulli	Villarreal	Goalkeeper	41	16	
2	3	Vlachodimos	Benfica	Goalkeeper	38	17	
3	4	Athanasiadis	Sheriff	Goalkeeper	29	8	
4	5	Mignolet	Club Brugge	Goalkeeper	28	20	
5	6	Oblak	Atlético	Goalkeeper	26	10	

	saved_penalties	cleansheets	punches made	match_played
0	1	5	4	13
1	0	3	12	12
2	1	5	5	10
3	0	1	2	5
4	0	0	1	6
5	0	3	2	10

TEAMSTATS.IPYNB

The final python notebook is the teamStats.ipynb notebook which compares all the teams that played with each other to find out which team performed the best

IMPORTING NECESSARY LIBRARIES

importing all the libraries required

+ Code + Markdown

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

[3] ✓ 0.0s Python

READING THE DATASETS

reading the csv datasets

```
df_disciplinary = pd.read_csv('archive/disciplinary.csv')
df_goals = pd.read_csv('archive/goals.csv')
fixtures_df = pd.read_csv('archive/fixtures.csv')
```

[21] ✓ 0.0s Python

PLOTTING A BAR GRAPH TO IDENTIFY THE NUMBER OF RED CARDS, YELLOW CARDS AND FOULS COMMITTED PER TEAM

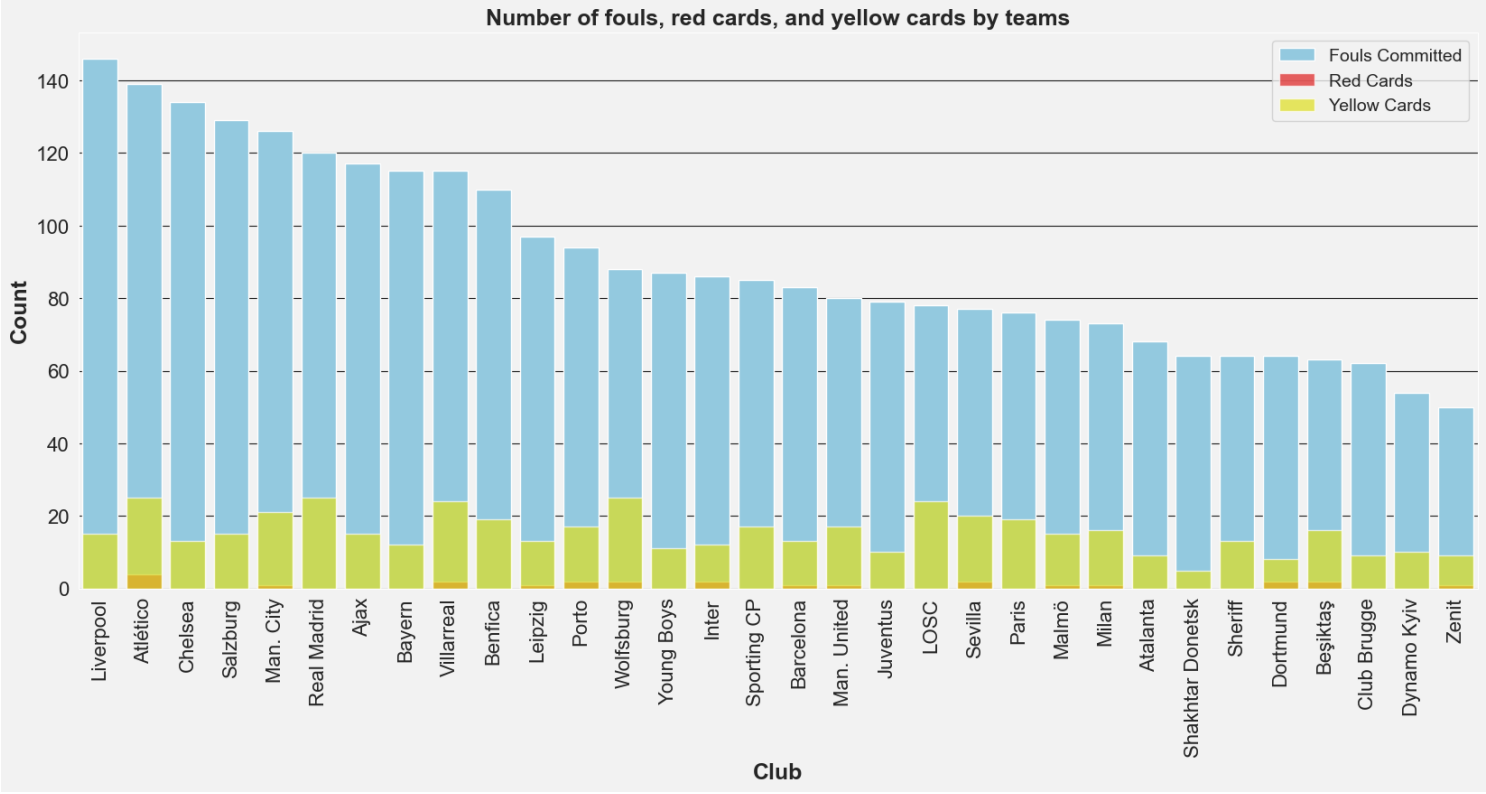
Number of fouls, red cards, and yellow cards by teams

+ Code + Markdown

```
plt.figure(figsize=(20, 8))
plt.title('Number of fouls, red cards, and yellow cards by teams', fontsize=18, fontweight='bold')
sns.set_style('darkgrid',
              {'axes.facecolor': '0.95',
               'grid.color': '0.1',
               'figure.facecolor': '0.95'})

# Grouping the data
grouped_data = df_disciplinary.groupby('club').sum()

# Sorting the values by fouls committed in descending order
sorted_data = grouped_data['fouls_committed'].sort_values(ascending=False)

# Plotting fouls committed
sns.barplot(x=sorted_data.index, y=sorted_data.values, color='skyblue', label='Fouls Committed')

# Plotting yellow cards (flipped with red cards)
sns.barplot(x=sorted_data.index, y=grouped_data.loc[sorted_data.index, 'yellow'], color='red', alpha=0.7, label='Red Cards')

# Plotting red cards (flipped with yellow cards)
sns.barplot(x=sorted_data.index, y=grouped_data.loc[sorted_data.index, 'red'], color='yellow', alpha=0.7, label='Yellow Cards')

plt.xlabel('Club', fontsize=18, fontweight='bold')
plt.ylabel('Count', fontsize=18, fontweight='bold')
plt.xticks(fontsize=16, rotation=90)
plt.yticks(fontsize=16)
plt.legend(fontsize=14)

plt.show()
```

[22] ✓ 1.4s Python

OUTPUT



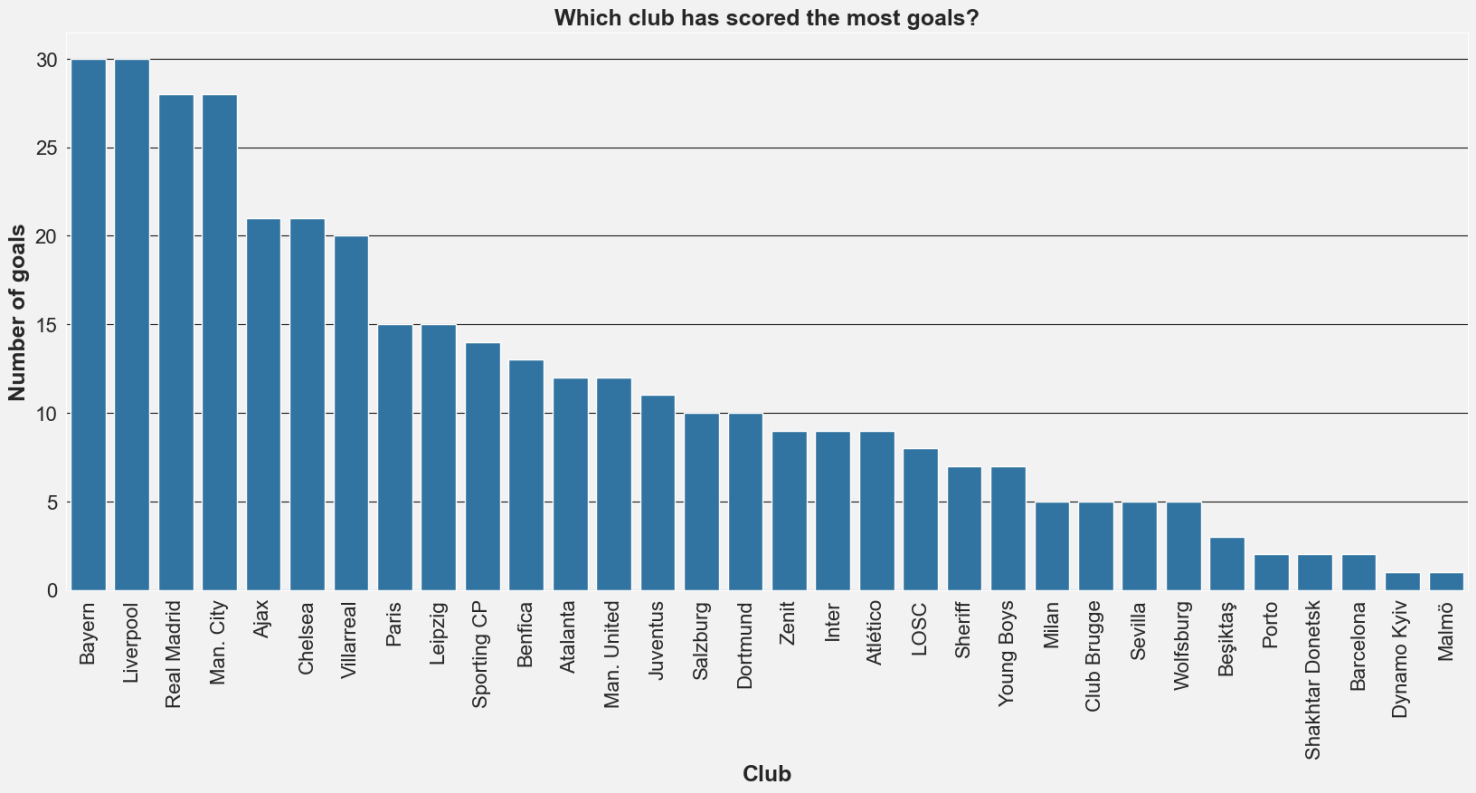
BAR GRAPH TO FIND THE GOALS SCORED BY THE CLUBS

Goals scored by all the clubs

```python
# Grouping the data
grouped_data = df_goals.groupby('club').sum()

# Sorting the values by goals scored in descending order
sorted_data = grouped_data['goals'].sort_values(ascending=False)

# Plotting the bar chart
plt.figure(figsize=(20, 8))
plt.title('Which club has scored the most goals?', fontsize=18, fontweight='bold')
plt.xticks(fontsize=16, rotation=90)
plt.yticks(fontsize=16)
sns.set_style('darkgrid',
              {'axes.facecolor': '0.95',
               'grid.color': '0.1',
               'figure.facecolor': '0.95'})
sns.barplot(x=sorted_data.index, y=sorted_data.values, order=sorted_data.index)
plt.xlabel('Club', fontsize=18, fontweight='bold')
plt.ylabel('Number of goals', fontsize=18, fontweight='bold')
plt.show()
```

[23] ✓ 0.6s Python

OUTPUT



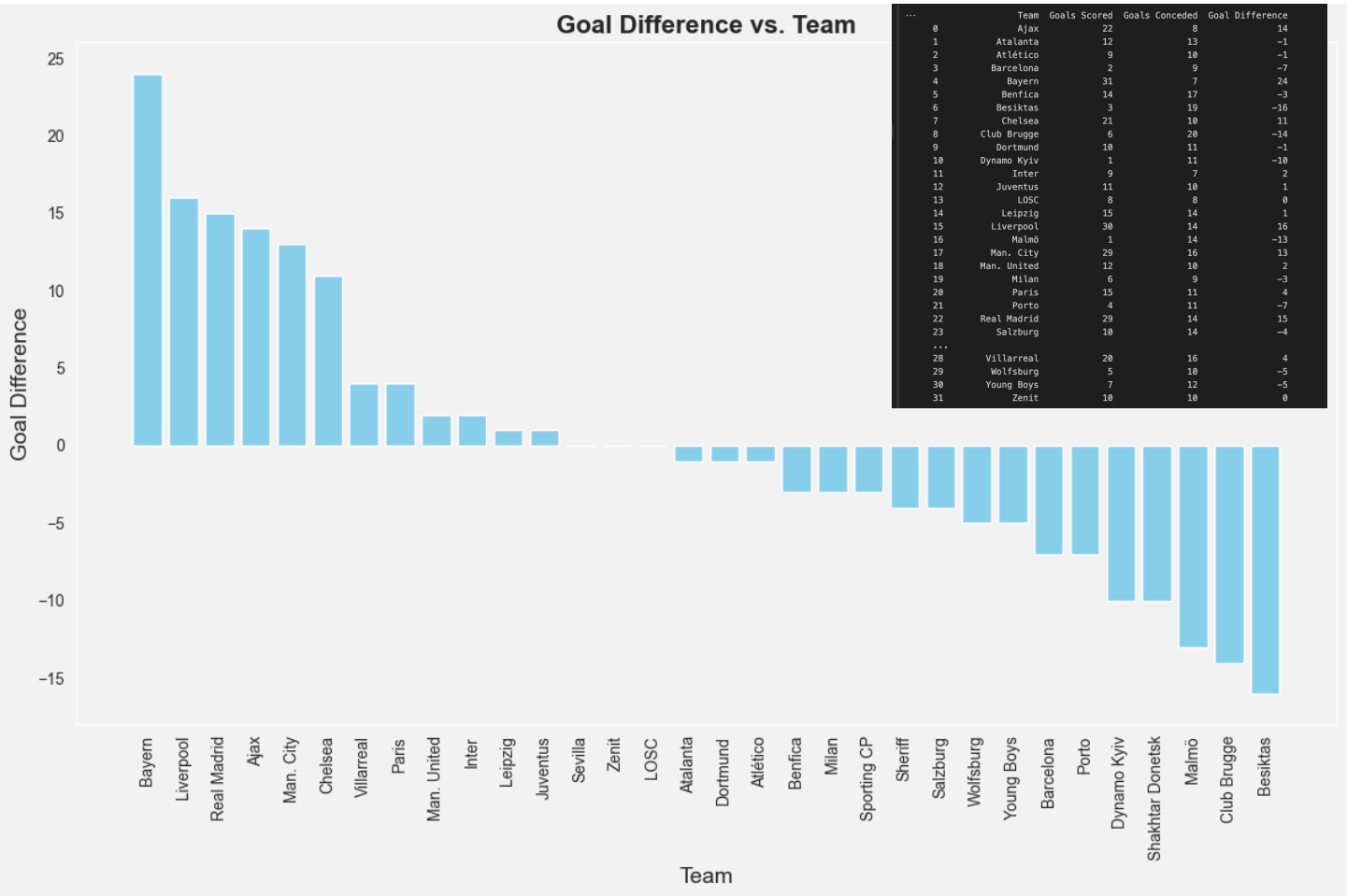
CALCULATING GOAL DIFFERENCE FOR ALL THE TEAMS

Goal difference for all the teams in the league

```
# Split the "Result" column into separate columns
fixtures_df[['Home Goals', 'Away Goals']] = fixtures_df['Result'].str.split(' - ', expand=True)

# Convert the columns to numeric types
fixtures_df['Home Goals'] = pd.to_numeric(fixtures_df['Home Goals'])
fixtures_df['Away Goals'] = pd.to_numeric(fixtures_df['Away Goals'])

# Calculate total goals scored and conceded by each team
goals_scored = fixtures_df.groupby('Home Team')['Home Goals'].sum().add(fixtures_df.groupby('Away Team')['Away Goals'].sum(), fill_value=0)
goals_conceded = fixtures_df.groupby('Home Team')['Away Goals'].sum().add(fixtures_df.groupby('Away Team')['Home Goals'].sum(), fill_value=0)

# Calculate goal difference
goal_difference = goals_scored - goals_conceded

# Create a DataFrame or list with team names, goals scored, goals conceded, and goal difference
team_stats_df = pd.DataFrame({
    'Team': goals_scored.index,
    'Goals Scored': goals_scored.values,
    'Goals Conceded': goals_conceded.values,
    'Goal Difference': goal_difference.values
})

# Display the team stats
print(team_stats_df)
```

[24] ✓ 0.0s Python

OUTPUT



	Team	Goals Scored	Goals Conceded	Goal Difference
0	Ajax	22	8	14
1	Atalanta	12	13	−1
2	Atlético	9	10	−1
3	Barcelona	2	9	−7
4	Bayern	31	7	24
5	Benfica	14	17	−3
6	Besiktas	3	19	−16
7	Chelsea	21	10	11
8	Club Brugge	6	20	−14
9	Dortmund	10	11	−1
10	Dynamo Kyiv	1	11	−10
11	Inter	9	7	2
12	Juventus	11	10	1
13	LOSC	8	8	0
14	Leipzig	15	14	1
15	Liverpool	30	14	16
16	Malmö	1	14	−13
17	Man. City	29	16	13
18	Man. United	12	10	2
19	Milan	6	9	−3
20	Paris	15	11	4
21	Porto	4	11	−7
22	Real Madrid	29	14	15
23	Salzburg	10	14	−4
...				
28	Villarreal	20	16	4
29	Wolfsburg	5	10	−5
30	Young Boys	7	12	−5
31	Zenit	10	10	0

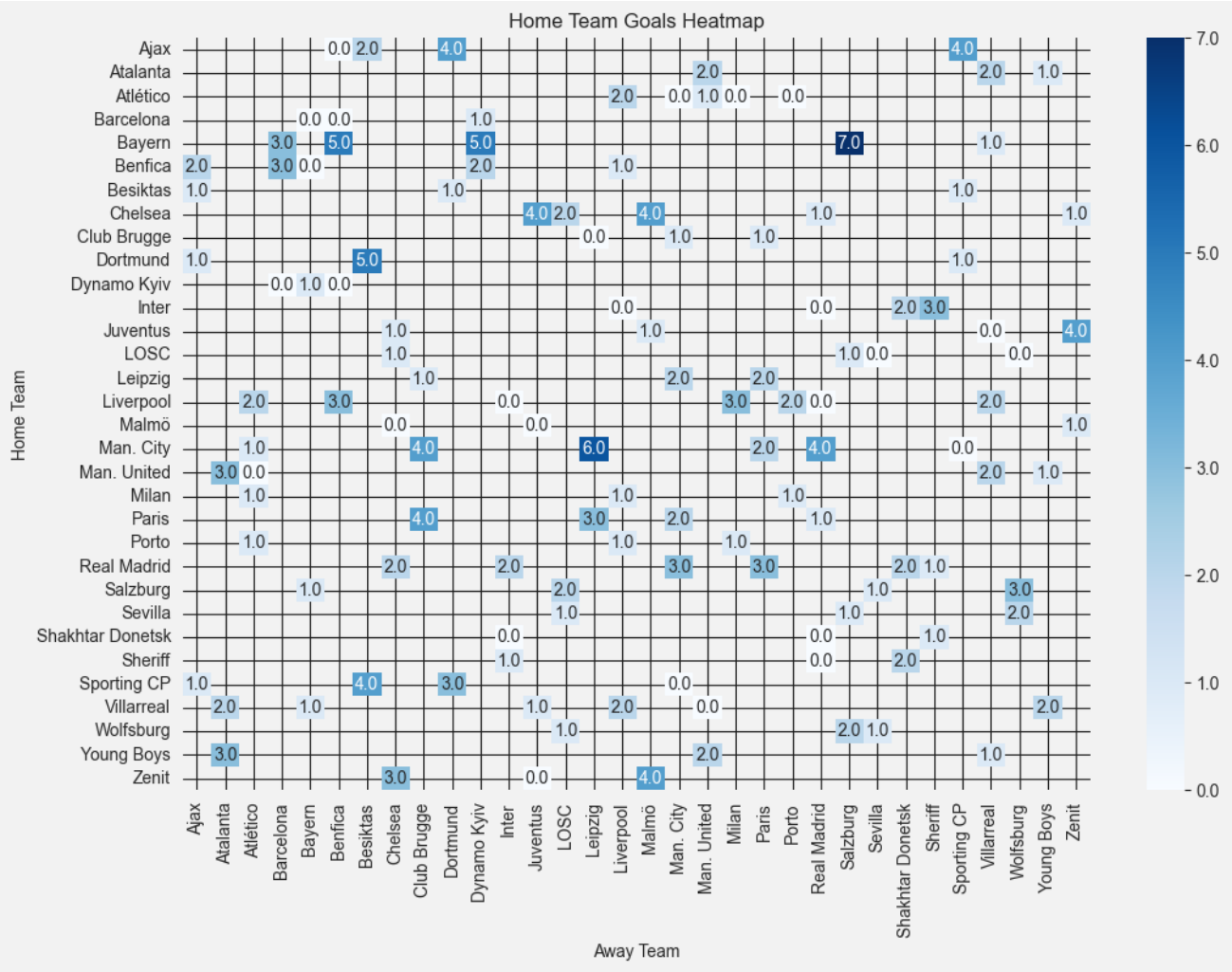
HEATMAP TO REPRESENT THE FIXTURES AND SCORELINES

Heatmap representing the scores and results between every team when the played each other

```
# Clean the 'Result' column to handle missing or invalid data
fixtures_df['Result'] = fixtures_df['Result'].fillna('0 - 0')  # Replace missing values with '0 - 0'
fixtures_df['Result'] = fixtures_df['Result'].str.replace(' ', '')  # Remove any spaces in the string

# Extract individual scores from the 'Result' column
fixtures_df[['Home Goals', 'Away Goals']] = fixtures_df['Result'].str.split('-', expand=True).astype(float)

# Create a pivot table with the home team as rows, away team as columns, and the result (score) as values
heatmap_data = fixtures_df.pivot_table(index='Home Team', columns='Away Team', values=['Home Goals', 'Away Goals'])

# Plot the pivot table as a heatmap with black background
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data['Home Goals'], cmap='Blues', annot=True, fmt='.1f', cbar_kws={'format': '%.1f'}, facecolor='black')
plt.title('Home Team Goals Heatmap')
plt.xlabel('Away Team')
plt.ylabel('Home Team')

plt.grid('gray')
plt.show()
```

[27] ✓ 1.0s Python

OUTPUT


Home Team Goals Heatmap
Home Team
Away Team
Ajax
Atalanta
Atlético
Barcelona
Bayern
Benfica
Besiktas
Chelsea
Club Brugge
Dortmund
Dynamo Kyiv
Inter
Juventus
LOSC
Leipzig
Liverpool
Malmö
Man. City
Man. United
Milan
Paris
Porto
Real Madrid
Salzburg
Sevilla
Shakhtar Donetsk
Sheriff
Sporting CP
Villarreal
Wolfsburg
Young Boys
Zenit
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0