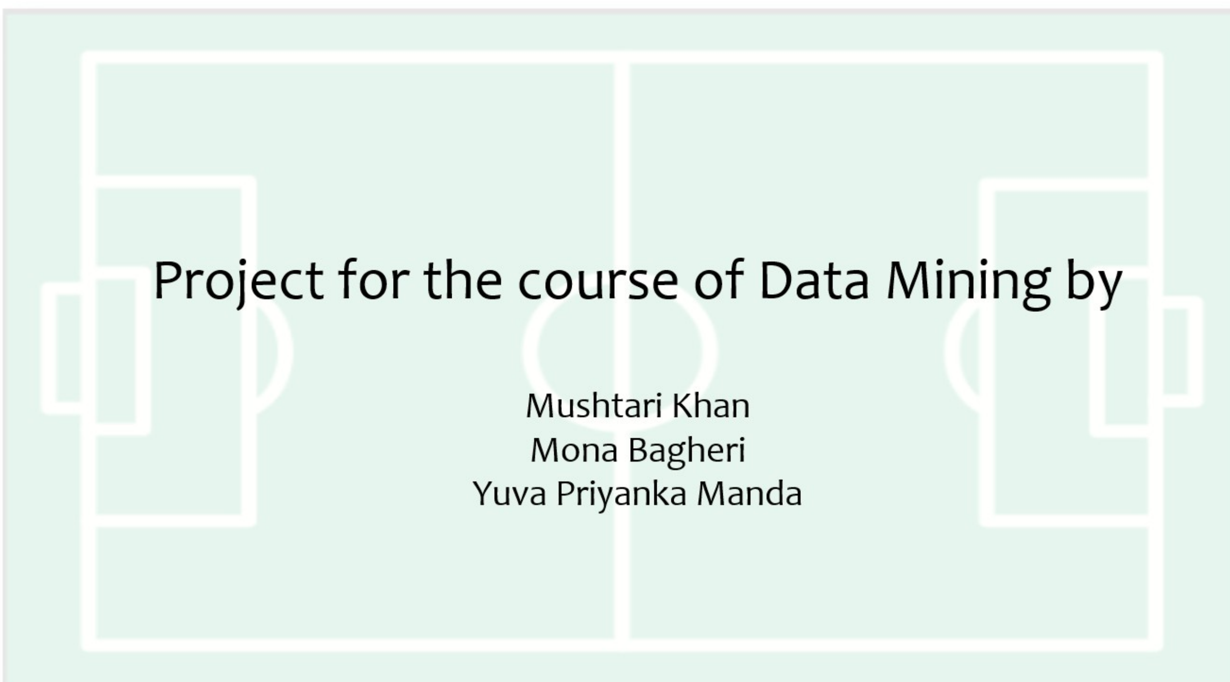
Introduction



Our project **"Exploring the patterns of European football and FIFA Players - A Data Mining approach"** explores the use of data mining concepts in the domain of Football. We have considered the dataset from Kaggle and appied various data mining techniques as part of this project.

Following a statistical approach, we applied techniques such as exploratory data analysis (EDA), clustering, hypothesis testing, and association rule mining to gain insights into player performance, team tactics, and match outcomes.

Importing Libraries

Importing all the necessary libraries for the project implementation.

```
import IPython
IPython.get_ipython().run_line_magic('config', 'IPKernelApp.matplotlib
= "inline"')

import warnings
warnings.filterwarnings('ignore')

import warnings
# Suppressing only DeprecationWarnings
warnings.filterwarnings('ignore', category=DeprecationWarning)

%matplotlib inline
import matplotlib.pyplot as plt
```

from google.colab import drive drive.mount('/content/drive')

```
pip install scikit-fuzzy mlxtend

Requirement already satisfied: scikit-fuzzy in
/home/unina/anaconda3/lib/python3.12/site-packages (0.5.0)
Requirement already satisfied: mlxtend in
/home/unina/anaconda3/lib/python3.12/site-packages (0.23.1)
Requirement already satisfied: scipy>=1.2.1 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(1.13.1)
Requirement already satisfied: numpy>=1.16.2 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(1.26.4)
Requirement already satisfied: pandas>=0.24.2 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(2.2.2)
Requirement already satisfied: scikit-learn>=1.0.2 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(1.4.2)
Requirement already satisfied: matplotlib>=3.0.0 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(3.8.4)
Requirement already satisfied: joblib>=0.13.2 in
/home/unina/anaconda3/lib/python3.12/site-packages (from mlxtend)
(1.4.2)
```

```
Requirement already satisfied: contourpy>=1.0.1 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (23.2)
Requirement already satisfied: pillow>=8 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
matplotlib>=3.0.0->mlxtend) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
/home/unina/anaconda3/lib/python3.12/site-packages (from
pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/unina/anaconda3/lib/python3.12/site-packages (from scikit-
learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in
/home/unina/anaconda3/lib/python3.12/site-packages (from python-
dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
!pip install MiniSom
```

```
Requirement already satisfied: MiniSom in
/home/unina/anaconda3/lib/python3.12/site-packages (2.3.3)
```

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
#import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import silhouette_score
```

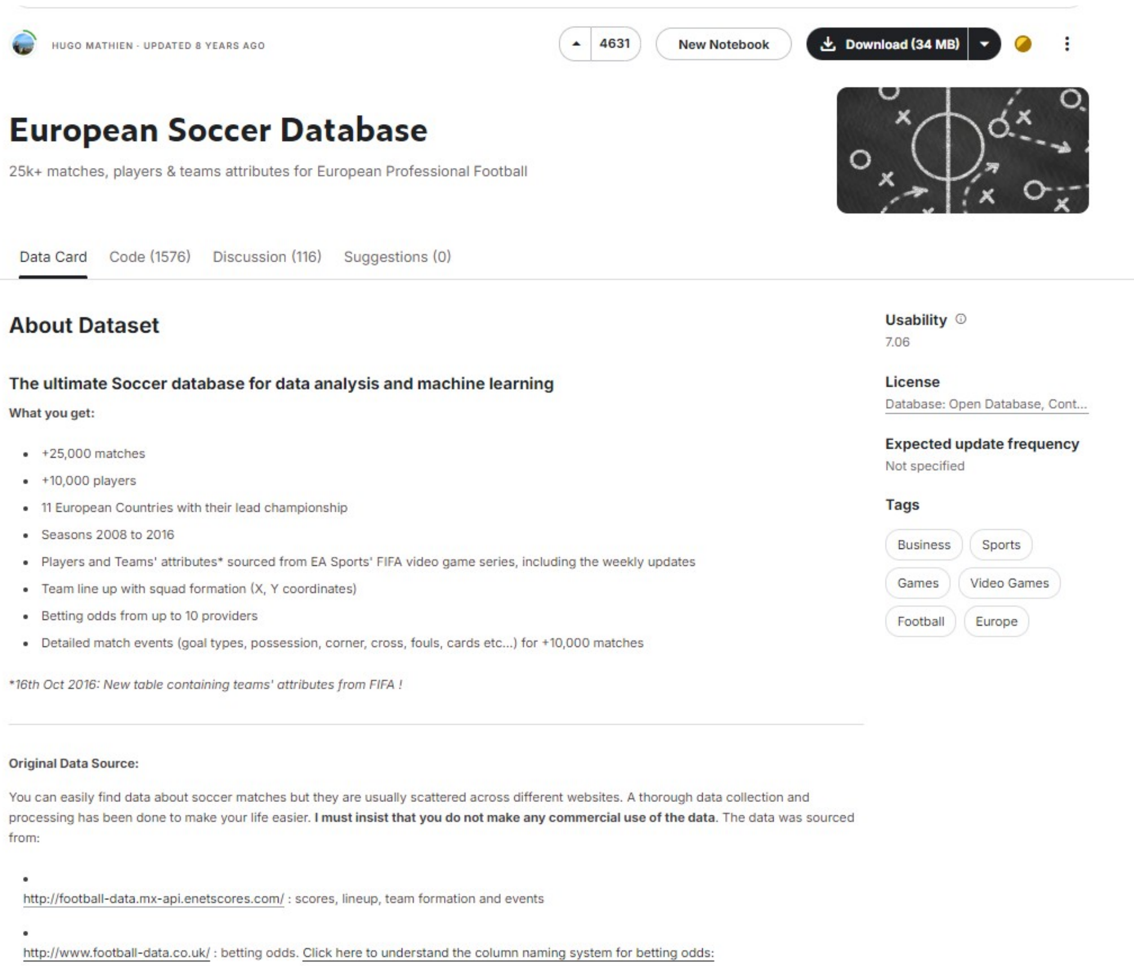
```python
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import skfuzzy as fuzz
import plotly.express as px
import scipy.stats as stats
from scipy import stats
import statsmodels.api as sm
import plotly.graph_objects as go
import plotly.figure_factory as ff
#from pgmpy.models import BayesianModel
#from pgmpy.factors.discrete import TabularCPD
#from pgmpy.inference import VariableElimination
from sklearn.decomposition import PCA
from statsmodels.multivariate.manova import MANOVA
from scipy.cluster.hierarchy import dendrogram, linkage
import seaborn as sns
from scipy.stats import skew
from scipy.stats import mannwhitneyu
from scipy.stats import chi2_contingency

#Kmeans
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA

#Univariate Analysis(Distribution)
import pandas as pd
import plotly.graph_objects as go
import plotly.figure_factory as ff
from scipy import stats

#Bivariate Analysis
import plotly.express as px

#Multivariate Analysis
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#Optimal n
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

```python
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

#K-means
from sklearn.decomposition import PCA
import plotly.express as px
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

#GMM
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
import plotly.express as px
from sklearn.decomposition import PCA

#Fuzzy C-Means
import numpy as np
import pandas as pd
import skfuzzy as fuzz
from sklearn.preprocessing import StandardScaler
import plotly.express as px
from sklearn.decomposition import PCA

#SOM
import numpy as np
import pandas as pd
from minisom import MiniSom
from sklearn.preprocessing import StandardScaler
import plotly.graph_objs as go
import plotly.express as px

#Association Rules
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
import networkx as nx
import matplotlib.pyplot as plt
```

Data Source

In this project, we analyze an extensive football dataset called European Soccer Database from Kaggle spanning 2008 to 2016, which includes data from 25,000+ matches and 10,000+ players across 11 European countries. The dataset provides rich information on player and team attributes, sourced from EA Sports' FIFA video game series, along with weekly updates. It also includes detailed match events such as goals, fouls, cards, and possession stats for 10,000+

matches. This comprehensive dataset enables us to explore various aspects of football performance, tactics, and match dynamics, making it an ideal resource for applying data mining techniques and machine learning models to uncover meaningful insights.



HUGO MATHIEN · UPDATED 8 YEARS AGO

4631 New Notebook Download (34 MB)

European Soccer Database

25k+ matches, players & teams attributes for European Professional Football

Data Card Code (1576) Discussion (116) Suggestions (0)

About Dataset

The ultimate Soccer database for data analysis and machine learning

What you get:

- +25,000 matches
- +10,000 players
- 11 European Countries with their lead championship
- Seasons 2008 to 2016
- Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly updates
- Team line up with squad formation (X, Y coordinates)
- Betting odds from up to 10 providers
- Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches

**16th Oct 2016: New table containing teams' attributes from FIFA !*

Original Data Source:

You can easily find data about soccer matches but they are usually scattered across different websites. A thorough data collection and processing has been done to make your life easier. **I must insist that you do not make any commercial use of the data.** The data was sourced from:

- http://football-data.mx-api.enetscores.com/ : scores, lineup, team formation and events
- http://www.football-data.co.uk/ : betting odds. Click here to understand the column naming system for betting odds:

Usability
7.06

License
Database: Open Database, Cont...

Expected update frequency
Not specified

Tags
Business Sports Games Video Games Football Europe

The data provider has compiled the dataset from collecting the data from various sources. We accessesd the origin of the data and compiled the following tables in the domain context understanding section.

Domain Context Understanding

To understand the dataset well and to gain domain knowledge, we compiled the following list of description for the variables in both Teams and Players Table in our dataset

Player Performance Attributes

Column Name	Description
`id`	Unique identifier for each player in the dataset.
`player_fifa_api_id`	Unique FIFA API identifier assigned to the player, used to retrieve player data from the FIFA database.

Column Name	Description
`player_api_id`	Unique identifier assigned to the player in the dataset, which may differ from the FIFA API ID.
`date`	Date when the player's data was recorded or last updated in the dataset.
`overall_rating`	Player's overall skill rating, typically ranging from 1 to 100, reflecting the player's general ability.
`potential`	The highest possible rating that the player can achieve in the future, indicating potential for improvement.
`preferred_foot`	Indicates the player's dominant foot, either "left" or "right," used for playing and shooting.
`defensive_work_rate`	Represents the player's tendency to work defensively during the match, classified as "high," "medium," or "low."
`crossing`	A numerical rating (usually out of 100) indicating the player's ability to cross the ball accurately to teammates.
`finishing`	A numerical rating indicating the player's ability to convert scoring opportunities into goals.
`heading_accuracy`	A rating indicating the player's proficiency in scoring goals using headers.
`short_passing`	A rating indicating the player's ability to execute short passes accurately.
`dribbling`	A rating indicating the player's skill in dribbling the ball past opponents.
`free_kick_accuracy`	A rating indicating the player's accuracy in taking free kicks.
`long_passing`	A rating indicating the player's ability to make accurate long passes.
`ball_control`	A rating indicating the player's skill in controlling the ball while dribbling or receiving passes.
`acceleration`	A rating indicating how quickly the player can reach top speed from a standstill.
`sprint_speed`	A rating indicating the player's maximum speed while sprinting.
`reactions`	A rating reflecting the player's ability to react quickly to game situations, such as intercepting passes or shots.
`shot_power`	A rating indicating the power behind the player’s shots on goal.
`stamina`	A rating indicating the player's ability to maintain performance throughout the game.
`strength`	A rating indicating the player's physical strength and ability to hold off opponents.
`long_shots`	A rating indicating the player's ability to score from long distances.
`aggression`	A rating indicating the player's tendency to challenge opponents and tackle aggressively.
`interceptions`	A rating indicating the player's ability to intercept passes and disrupt the opposing team's play.
`positioning`	A rating indicating the player's ability to position themselves effectively

Column Name	Description
	during a match.
`penalties`	A rating indicating the player's proficiency in taking penalty kicks.
`marking`	A rating indicating the player's ability to mark opponents defensively.
`standing_tackle`	A rating indicating the player's ability to execute standing tackles effectively.
`gk_diving`	A rating for goalkeepers indicating their ability to dive to make saves.
`gk_handling`	A rating for goalkeepers indicating their skill in catching and holding the ball.
`gk_kicking`	A rating for goalkeepers indicating their ability to kick the ball accurately over long distances.
`gk_positioning`	A rating for goalkeepers indicating their ability to position themselves correctly during plays.
`gk_reflexes`	A rating for goalkeepers indicating their quickness and ability to react to shots.
`player_name`	The name of the player.

Team Attributes

Column Name	Description
id_x	A unique identifier for each team in the dataset, serving as the primary key for team data.
team_api_id	A unique identifier for each team used by the FIFA API for referencing team-related data across various datasets.
team_long_name	The full name of the team (e.g., "Manchester United Football Club").
team_short_name	The abbreviated or commonly used name of the team (e.g., "Man Utd" or "MUFC").
id_y	An alternative unique identifier for referencing the team, potentially from a different data source.
team_fifa_api_id	A unique identifier assigned to each team by the FIFA API for cross-referencing and data integrity.
date	The date when the team's attributes were recorded or updated, important for tracking changes over time.
buildUpPlaySpeed	A numerical rating measuring the speed at which a team builds up play from defense to attack (0 to 100 scale).
buildUpPlaySpeedClass	Categorical representation of `buildUpPlaySpeed` (e.g., "low," "medium," "high").
buildUpPlayDribblingClass	Categorical rating of a team’s dribbling effectiveness during the build-up phase.
buildUpPlayPassing	A numerical rating assessing a team's effectiveness in passing during the build-up phase.
buildUpPlayPa	Categorical representation of `buildUpPlayPassing` (e.g., "low,"

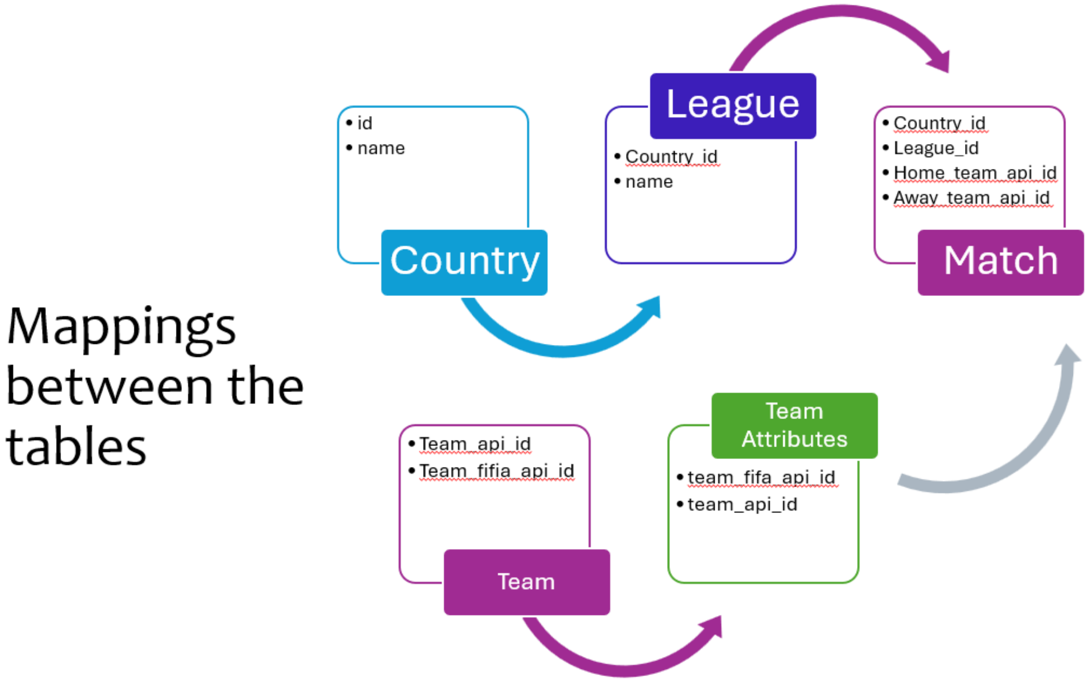
Column Name	Description
ssingClass	"medium," "high").
buildUpPlayPositioningClass	Categorical assessment of how well a team positions itself during the build-up phase.
chanceCreationPassing	A numerical rating indicating how effective a team is in creating chances through passing during the attacking phase.
chanceCreationPassingClass	Categorical representation of `chanceCreationPassing` (e.g., "low," "medium," "high").
chanceCreationCrossing	A numerical rating measuring a team's effectiveness in creating goal-scoring opportunities through crosses.
chanceCreationCrossingClass	Categorical representation of `chanceCreationCrossing` (e.g., "low," "medium," "high").
chanceCreationShooting	A numerical rating assessing how well a team creates goal-scoring opportunities through shooting efforts.
chanceCreationShootingClass	Categorical representation of `chanceCreationShooting` (e.g., "low," "medium," "high").
chanceCreationPositioningClass	Categorical assessment of how well a team positions itself to create scoring chances.
defencePressure	A numerical rating indicating how effectively a team applies pressure on the opposing team when they have possession.
defencePressureClass	Categorical representation of `defencePressure` (e.g., "low," "medium," "high").
defenceAggression	A numerical rating reflecting how aggressive a team is in their defensive actions, including tackles and interceptions.
defenceAggressionClass	Categorical representation of `defenceAggression` (e.g., "low," "medium," "high").
defenceTeamWidth	A numerical measure indicating how wide or narrow a team plays during defensive phases.
defenceTeamWidthClass	Categorical representation of `defenceTeamWidth` (e.g., "narrow," "normal," "wide").
defenceDefenderLineClass	Categorical assessment of the positioning of a team’s defensive line, classified based on height (e.g., "high," "medium," "low").

Table Mappings

The Dataset has 5 tables that are related to each other by foreign keys. And were in sqlite table format. The tables were converted to CSV format from SQL as a preprocessing step in an another colab file. For the sake of simplicity we directly import the CSV files in our project.

The code for SQL to CSV conversion is in the file named : SQL to CSV.ipynb

For a visual representation of how tables are associated we can see the below diagram



Objectives

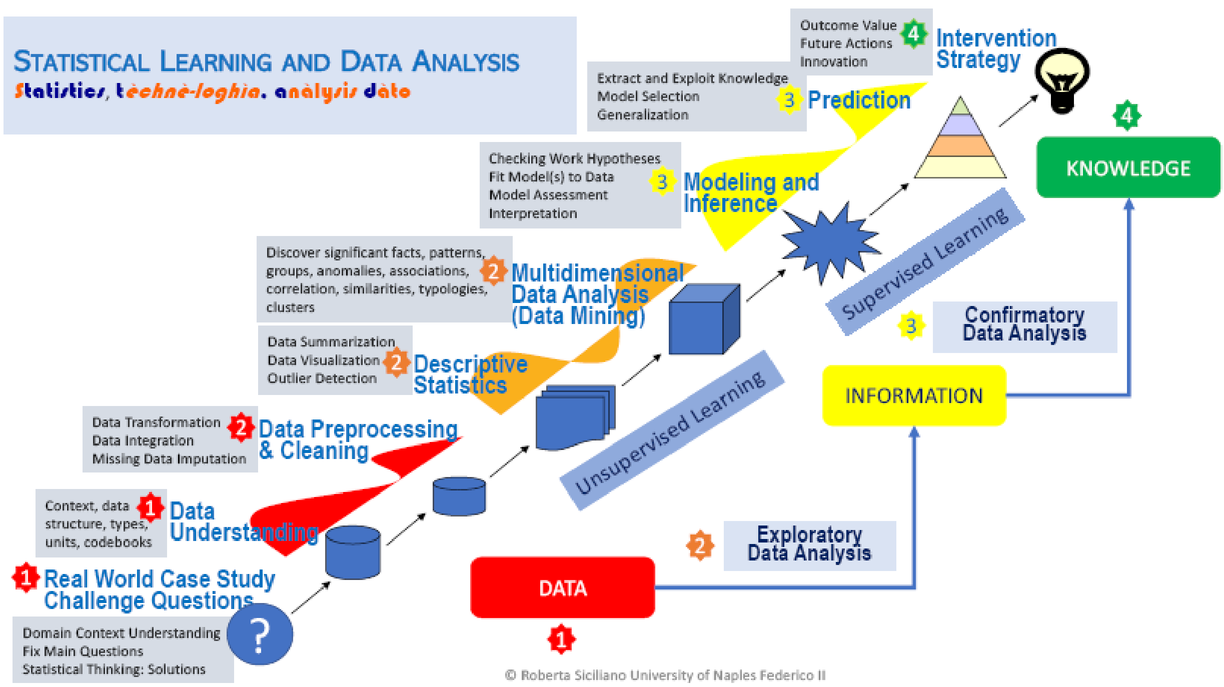
The Objectives of our project are:

- To perform comprehensive exploratory data analysis (EDA) and data preprocessing to understand the dataset, handle missing values, and prepare the data for analysis through feature engineering and transformation.
- To apply various Hypothesis tests to perform tests such as the t-test, Mann-Whitney U test, and Chi-square test, we aim to determine whether significant differences or associations exist between specific groups
- To apply various clustering techniques like K-Means, Fuzzy C-Means, GMM)to group players based on their attributes and identify distinct player archetypes, along with visualizing the results using dimensionality reduction techniques like PCA.
- To apply association rule mining techniques to uncover frequent patterns in player and team attributes, identifying how various player skills and team strategies are related to each other.

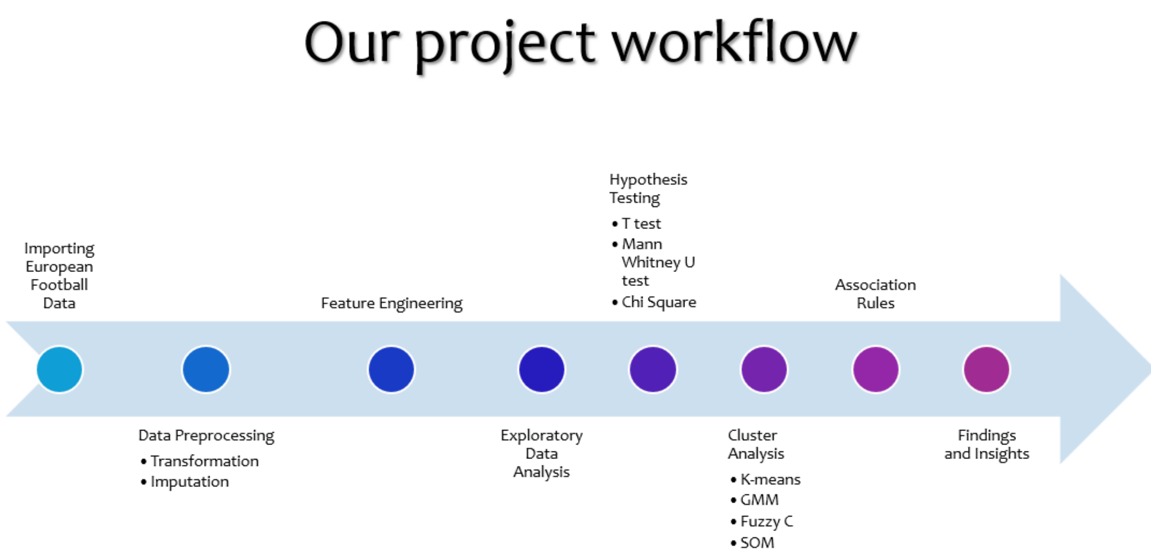
Methodology

To develop this project we follow a structured statistical learning and data analysis pipeline to guide our methodology. Starting with data understanding, we explore the dataset's structure and define key questions. We then proceed with data preprocessing and cleaning to handle missing values, standardize features, and prepare the data for analysis. Using descriptive statistics and exploratory data analysis (EDA), we uncover patterns, relationships, and outliers. Moving into the core of the project, we apply multidimensional data analysis (data mining) techniques, including clustering and association rule mining, to discover hidden patterns.

Statistical Approach Pipeline



Workflow of the Project



Statistical Questions

The following tables list the statistical questions we considered for the project. This approach of exploring the data and applying data mining concepts gave us insights and helped us to focus the analysis of our interest and ensured relevance to the data that lead us to more maningful insights.

	Plot Type	Plot Title	Statistical Question
Univariate	Bar Plot	Mean Overal Rating by age group	What age group of players have high overall rating
		Total Value in Euros by Age group	What age group of players have highest 'value euro' (overall sum)?
		Distribution of BMI Categories	What BMI categories the players fall into?
		Count of Players by Offensive Score Category	How many players have high Offensive Score Category
		Count of Players by Defensive Score Category	What category of defensive score most players fall into?
		Match Count by League	How many matched were played per each league?
		Match Result Distribution	What is the most frequent match outcome (Home Win, Away Win, or Draw) based on the match result distribution?
		Boxplot of Defence Pressure	What is the typical range of "Defence Pressure" values?
		Comparison of Home vs Away Team Goals	What is the typical goal distribution for home and away teams?
	Distribution	Distribution of Goal Difference (Home vs Away)	How does the goal difference vary between home and away matches?
		Distribution of Home Team Goals	How are the goals distributed by home teams in matches
		Distribution of Away Team Goals	How are the goals distributed by away teams in matches?
		Distribution of Build Up Play Speed Distribution of Build Up Play Passing Distribution of Chance Creation Passing Distribution of Defence Pressure Distribution of Defence Aggression Distribution of Defence Team Width Distribution of Chance Creation Shooting Distribution of Build Up Play Positioning	What is the distribution of build-up play speed among teams, and what is the typical speed achieved? How do teams' passing attributes vary in build-up play, and what is the central tendency of this passing metric? What is the distribution of chance creation passing among teams, and how frequently do teams achieve higher passing metrics? How is the defence pressure distributed across teams, and what are the common values for this attribute? What is the range of defensive aggression scores among teams, and do most teams exhibit high or low aggression? How does team width in defence vary across teams, and what are the most frequently observed widths? What is the distribution of chance creation shooting among teams, and how does this influence their scoring opportunities? How does the positioning during build-up play vary among teams, and what are the common positioning scores?
		Distribution of Finishing Attribute	What is the distribution of players' finishing skills, and how does it vary across different skill levels?
Bivariate	Groupped bar plot	Match result by stage	How are match results distributed across different stages of the season?
	scatter plot	Overall Rating vs. Value in Euros	How does a player's overall rating correlate with their market value in euros?
	box plot	Overall Rating by Preferred Foot	How does a player's overall rating vary depending on their preferred foot?
	Bar plot	Home vs Away Goals by league	How do average home goals compare to average away goals across different leagues?
	box plot	Goals Distribution by Country and League	How do the total goals differ across countries and leagues?
	Interactive Line Plot with dropdown menu	Home vs Away Team Performance by Season and League	How do the average home and away goals scored by teams vary across different seasons and leagues?
			How does the average total number of

Importing Tables

As stated earlier we import the data from csv format tables.

```
Country = 'CSV_Files/Country.csv'

# Reading  the CSV file
Country = pd.read_csv(Country, encoding='latin-1')

# Printing the first few rows of the DataFrame
print(Country.head())
```

```
      id     name
0      1  Belgium
1   1729  England
2   4769   France
3   7809  Germany
4  10257    Italy
```

```
League = 'CSV_Files/League.csv'

# Reading the CSV file
League = pd.read_csv(League, encoding='latin-1')

# Printing the first few rows of the DataFrame
print(League.head())
```

```
      id  country_id                    name
0      1           1  Belgium Jupiler League
1   1729        1729  England Premier League
2   4769        4769          France Ligue 1
3   7809        7809   Germany 1. Bundesliga
4  10257       10257           Italy Serie A
```

```
Match = 'CSV_Files/Match.csv'

# Reading the CSV file
Match = pd.read_csv(Match, encoding='latin-1')

# Printing the first few rows of the DataFrame
print(Match.head())
```

```
   id  country_id  league_id     season  stage                 date  \
0   1           1          1  2008/2009      1  2008-08-17 00:00:00
1   2           1          1  2008/2009      1  2008-08-16 00:00:00
2   3           1          1  2008/2009      1  2008-08-16 00:00:00
3   4           1          1  2008/2009      1  2008-08-17 00:00:00
4   5           1          1  2008/2009      1  2008-08-16 00:00:00

   match_api_id  home_team_api_id  away_team_api_id
home_team_goal  ...  \
```

```
0         492473               9987               9993
1  ...
1         492474              10000               9994
0  ...
2         492475               9984               8635
0  ...
3         492476               9991               9998
5  ...
4         492477               7947               9985
1  ...

    SJA   VCH   VCD   VCA   GBH   GBD   GBA   BSH   BSD   BSA
0  4.00  1.65  3.40  4.50  1.78  3.25  4.00  1.73  3.40  4.20
1  3.80  2.00  3.25  3.25  1.85  3.25  3.75  1.91  3.25  3.60
2  2.50  2.35  3.25  2.65  2.50  3.20  2.50  2.30  3.20  2.75
3  7.50  1.45  3.75  6.50  1.50  3.75  5.50  1.44  3.75  6.50
4  1.73  4.50  3.40  1.65  4.50  3.50  1.65  4.75  3.30  1.67

[5 rows x 115 columns]
```

```
Team = 'CSV_Files/Team.csv'

# Reading the CSV file
Team = pd.read_csv(Team, encoding='latin-1')

# Printing the first few rows of the DataFrame
print(Team.head())
```

```
   id  team_api_id  team_fifa_api_id     team_long_name
team_short_name
0   1         9987             673.0           KRC Genk
GEN
1   2         9993             675.0       Beerschot AC
BAC
2   3        10000           15005.0   SV Zulte-Waregem
ZUL
3   4         9994            2007.0   Sporting Lokeren
LOK
4   5         9984            1750.0  KSV Cercle Brugge
CEB
```

```
Team_Attributes = 'CSV_Files/Team_Attributes.csv'

# Reading the CSV file
Team_Attributes = pd.read_csv(Team_Attributes, encoding='latin-1')

# Printing the first few rows of the DataFrame
print(Team_Attributes.head())
```

```
   id  team_fifa_api_id  team_api_id                 date
buildUpPlaySpeed  \
```

```
0   1               434         9930  2010-02-22 00:00:00
60
1   2               434         9930  2014-09-19 00:00:00
52
2   3               434         9930  2015-09-10 00:00:00
47
3   4                77         8485  2010-02-22 00:00:00
70
4   5                77         8485  2011-02-22 00:00:00
47

  buildUpPlaySpeedClass  buildUpPlayDribbling
buildUpPlayDribblingClass  \
0              Balanced                   NaN
Little
1              Balanced                  48.0
Normal
2              Balanced                  41.0
Normal
3                  Fast                   NaN
Little
4              Balanced                   NaN
Little

   buildUpPlayPassing buildUpPlayPassingClass  ...
chanceCreationShooting  \
0                  50                   Mixed  ...
55
1                  56                   Mixed  ...
64
2                  54                   Mixed  ...
64
3                  70                    Long  ...
70
4                  52                   Mixed  ...
52

   chanceCreationShootingClass chanceCreationPositioningClass  \
0                       Normal                      Organised
1                       Normal                      Organised
2                       Normal                      Organised
3                         Lots                      Organised
4                       Normal                      Organised

   defencePressure defencePressureClass  defenceAggression  \
0               50               Medium                 55
1               47               Medium                 44
2               47               Medium                 44
3               60               Medium                 70
4               47               Medium                 47
```

```
  defenceAggressionClass defenceTeamWidth  defenceTeamWidthClass  \
0                  Press               45                 Normal
1                  Press               54                 Normal
2                  Press               54                 Normal
3                 Double               70                   Wide
4                  Press               52                 Normal

  defenceDefenderLineClass
0                    Cover
1                    Cover
2                    Cover
3                    Cover
4                    Cover

[5 rows x 25 columns]

# Defining the path to the CSV file
data = 'CSV_Files/fifa_players.csv'

# Loading the CSV file
Player_Attributes_with_Names = pd.read_csv(data)

# Displaying the first few rows of the DataFrame
Player_Attributes_with_Names.head()

           name                          full_name birth_date  age
height_cm  \
0      L. Messi  Lionel Andrés Messi Cuccittini  6/24/1987   31
170.18
1    C. Eriksen    Christian  Dannemann Eriksen  2/14/1992   27
154.94
2      P. Pogba                        Paul Pogba  3/15/1993   25
190.50
3    L. Insigne                   Lorenzo Insigne   6/4/1991   27
162.56
4  K. Koulibaly                Kalidou Koulibaly  6/20/1991   27
187.96

   weight_kgs  positions nationality  overall_rating
potential  ...  \
0        72.1   CF,RW,ST   Argentina              94          94  ...

1        76.2  CAM,RM,CM     Denmark              88          89  ...

2        83.9     CM,CAM      France              88          91  ...

3        59.0      LW,ST       Italy              88          88  ...

4        88.9         CB     Senegal              88          91  ...
```

```
   long_shots  aggression interceptions  positioning  vision
penalties  \
0          94          48            22           94      94
75
1          89          46            56           84      91
67
2          82          78            64           82      88
82
3          84          34            26           83      87
61
4          15          87            88           24      49
33

  composure  marking standing_tackle  sliding_tackle
0        96       33              28              26
1        88       59              57              22
2        87       63              67              67
3        83       51              24              22
4        80       91              88              87

[5 rows x 51 columns]
```

Preprocessing

As part of preprocessing we performed data transformation to the tables by merging the tables by their associations as listed in the Table Mappings section. This provided us with comprehensive tables that were further used for data mining. Imputation was also a major part of the preprocessing section by which we handled the missing values in the dataset.

Team and Team attributes Table

```
Team_Attributes.shape

(1458, 25)

Team_Attributes.columns.tolist()

['id',
 'team_fifa_api_id',
 'team_api_id',
 'date',
 'buildUpPlaySpeed',
 'buildUpPlaySpeedClass',
 'buildUpPlayDribbling',
 'buildUpPlayDribblingClass',
 'buildUpPlayPassing',
 'buildUpPlayPassingClass',
 'buildUpPlayPositioningClass',
```

```
 'chanceCreationPassing',
 'chanceCreationPassingClass',
 'chanceCreationCrossing',
 'chanceCreationCrossingClass',
 'chanceCreationShooting',
 'chanceCreationShootingClass',
 'chanceCreationPositioningClass',
 'defencePressure',
 'defencePressureClass',
 'defenceAggression',
 'defenceAggressionClass',
 'defenceTeamWidth',
 'defenceTeamWidthClass',
 'defenceDefenderLineClass']
```

```
Team.shape
```

```
(299, 5)
```

```
Team.columns.tolist()
```

```
['id', 'team_api_id', 'team_fifa_api_id', 'team_long_name',
'team_short_name']
```

Country, Match League Tables

```
Match.shape
```

```
(25979, 115)
```

```
Match.columns.tolist()
```

```
['id',
 'country_id',
 'league_id',
 'season',
 'stage',
 'date',
 'match_api_id',
 'home_team_api_id',
 'away_team_api_id',
 'home_team_goal',
 'away_team_goal',
 'home_player_X1',
 'home_player_X2',
 'home_player_X3',
 'home_player_X4',
 'home_player_X5',
 'home_player_X6',
 'home_player_X7',
 'home_player_X8',
```

```
'home_player_X9',
'home_player_X10',
'home_player_X11',
'away_player_X1',
'away_player_X2',
'away_player_X3',
'away_player_X4',
'away_player_X5',
'away_player_X6',
'away_player_X7',
'away_player_X8',
'away_player_X9',
'away_player_X10',
'away_player_X11',
'home_player_Y1',
'home_player_Y2',
'home_player_Y3',
'home_player_Y4',
'home_player_Y5',
'home_player_Y6',
'home_player_Y7',
'home_player_Y8',
'home_player_Y9',
'home_player_Y10',
'home_player_Y11',
'away_player_Y1',
'away_player_Y2',
'away_player_Y3',
'away_player_Y4',
'away_player_Y5',
'away_player_Y6',
'away_player_Y7',
'away_player_Y8',
'away_player_Y9',
'away_player_Y10',
'away_player_Y11',
'home_player_1',
'home_player_2',
'home_player_3',
'home_player_4',
'home_player_5',
'home_player_6',
'home_player_7',
'home_player_8',
'home_player_9',
'home_player_10',
'home_player_11',
'away_player_1',
'away_player_2',
```

```
 'away_player_3',
 'away_player_4',
 'away_player_5',
 'away_player_6',
 'away_player_7',
 'away_player_8',
 'away_player_9',
 'away_player_10',
 'away_player_11',
 'goal',
 'shoton',
 'shotoff',
 'foulcommit',
 'card',
 'cross',
 'corner',
 'possession',
 'B365H',
 'B365D',
 'B365A',
 'BWH',
 'BWD',
 'BWA',
 'IWH',
 'IWD',
 'IWA',
 'LBH',
 'LBD',
 'LBA',
 'PSH',
 'PSD',
 'PSA',
 'WHH',
 'WHD',
 'WHA',
 'SJH',
 'SJD',
 'SJA',
 'VCH',
 'VCD',
 'VCA',
 'GBH',
 'GBD',
 'GBA',
 'BSH',
 'BSD',
 'BSA']
```

```
League.shape
```

```
(11, 3)
```

```
League.columns.tolist()
```

```
['id', 'country_id', 'name']
```

```
Country.shape
```

```
(11, 2)
```

```
Country.columns.tolist()
```

```
['id', 'name']
```

Players Table

```
Player_Attributes_with_Names.shape
```

```
(17954, 51)
```

```
Player_Attributes_with_Names.columns.tolist()
```

```
['name',
 'full_name',
 'birth_date',
 'age',
 'height_cm',
 'weight_kgs',
 'positions',
 'nationality',
 'overall_rating',
 'potential',
 'value_euro',
 'wage_euro',
 'preferred_foot',
 'international_reputation(1-5)',
 'weak_foot(1-5)',
 'skill_moves(1-5)',
 'body_type',
 'release_clause_euro',
 'national_team',
 'national_rating',
 'national_team_position',
 'national_jersey_number',
 'crossing',
 'finishing',
 'heading_accuracy',
 'short_passing',
 'volleys',
 'dribbling',
 'curve',
```

```
 'freekick_accuracy',
 'long_passing',
 'ball_control',
 'acceleration',
 'sprint_speed',
 'agility',
 'reactions',
 'balance',
 'shot_power',
 'jumping',
 'stamina',
 'strength',
 'long_shots',
 'aggression',
 'interceptions',
 'positioning',
 'vision',
 'penalties',
 'composure',
 'marking',
 'standing_tackle',
 'sliding_tackle']
```

Data Transformation

Merging of the Tables

1. Team Table with Team Attributes
2. Country, League , Match table with Team

```
#Merging the DataFrames on 'team_api_id'
Team_Attributes_with_Names = pd.merge(Team, Team_Attributes,
on='team_api_id', how='inner')

#Reviewing the merged DataFrame
print(Team_Attributes_with_Names.head())

   id_x  team_api_id  team_fifa_api_id_x team_long_name
team_short_name  id_y  \
0      1         9987                673.0        KRC Genk
GEN   485
1      1         9987                673.0        KRC Genk
GEN   486
2      1         9987                673.0        KRC Genk
GEN   487
3      1         9987                673.0        KRC Genk
GEN   488
4      1         9987                673.0        KRC Genk
GEN   489
```

```
   team_fifa_api_id_y                 date  buildUpPlaySpeed  \
0                 673  2010-02-22 00:00:00                45   
1                 673  2011-02-22 00:00:00                66   
2                 673  2012-02-22 00:00:00                53   
3                 673  2013-09-20 00:00:00                58   
4                 673  2014-09-19 00:00:00                58   

  buildUpPlaySpeedClass  ...  chanceCreationShooting  \
0              Balanced  ...                      60   
1              Balanced  ...                      51   
2              Balanced  ...                      56   
3              Balanced  ...                      56   
4              Balanced  ...                      56   

  chanceCreationShootingClass  chanceCreationPositioningClass 
defencePressure  \
0                      Normal                       Organised 
70   
1                      Normal                       Organised 
48   
2                      Normal                       Organised 
47   
3                      Normal                       Organised 
47   
4                      Normal                       Organised 
47   

  defencePressureClass  defenceAggression defenceAggressionClass  \
0                 High                 65                  Press   
1               Medium                 47                  Press   
2               Medium                 45                  Press   
3               Medium                 45                  Press   
4               Medium                 45                  Press   

   defenceTeamWidth defenceTeamWidthClass  defenceDefenderLineClass  
0                70                  Wide                     Cover  
1                54                Normal              Offside Trap  
2                55                Normal                     Cover  
3                55                Normal                     Cover  
4                55                Normal                     Cover  

[5 rows x 29 columns]
```

Mappings to be Done:

1. country_id in League Table
2. Coutry id and league id in Match table
3. Home team api id in Match table
4. Away team api id in match table

```python
#Merging Country and League tables on country_id
country_league_df = pd.merge(League, Country, left_on='country_id',
right_on='id', suffixes=('_league', '_country'))

#Merging the result with Match table on league_id and country_id
Merged_LMC = pd.merge(Match, country_league_df, left_on=['league_id',
'country_id'], right_on=['id_league', 'country_id'], how='left')

# Dropping redundant columns after merging
Merged_LMC = Merged_LMC.drop(columns=['id_league', 'id_country'])
```

Combining Team with Match, Country, League Table

```python
# Merging for Home Team Details
merged_with_home = pd.merge(
    Merged_LMC,
    Team[['team_api_id', 'team_long_name', 'team_short_name']],
    left_on='home_team_api_id',
    right_on='team_api_id',
    how='left'
)

# Renaming columns to reflect they are for the home team
merged_with_home.rename(columns={
    'team_long_name': 'home_team_long_name',
    'team_short_name': 'home_team_short_name'
}, inplace=True)

# Dropping the extra team_api_id column from the first merge
merged_with_home.drop('team_api_id', axis=1, inplace=True)

# Merging for Away Team Details
MCLT_Combined = pd.merge(
    merged_with_home,
    Team[['team_api_id', 'team_long_name', 'team_short_name']],
    left_on='away_team_api_id',
    right_on='team_api_id',
    how='left'
)

# Renaming columns to reflect they are for the away team
MCLT_Combined.rename(columns={
    'team_long_name': 'away_team_long_name',
    'team_short_name': 'away_team_short_name'
}, inplace=True)

# Dropping the extra team_api_id column from the second merge
MCLT_Combined.drop('team_api_id', axis=1, inplace=True)
```

```python
# Printing the first few rows of the DataFrame
print(MCLT_Combined.head())
```

```
   id  country_id  league_id     season  stage                 date  \
0   1           1          1  2008/2009      1  2008-08-17 00:00:00
1   2           1          1  2008/2009      1  2008-08-16 00:00:00
2   3           1          1  2008/2009      1  2008-08-16 00:00:00
3   4           1          1  2008/2009      1  2008-08-17 00:00:00
4   5           1          1  2008/2009      1  2008-08-16 00:00:00

   match_api_id  home_team_api_id  away_team_api_id
home_team_goal  ...  \
0        492473              9987              9993
1  ...
1        492474             10000              9994
0  ...
2        492475              9984              8635
0  ...
3        492476              9991              9998
5  ...
4        492477              7947              9985
1  ...

    GBA   BSH   BSD   BSA             name_league  name_country  \
0  4.00  1.73  3.40  4.20  Belgium Jupiler League       Belgium
1  3.75  1.91  3.25  3.60  Belgium Jupiler League       Belgium
2  2.50  2.30  3.20  2.75  Belgium Jupiler League       Belgium
3  5.50  1.44  3.75  6.50  Belgium Jupiler League       Belgium
4  1.65  4.75  3.30  1.67  Belgium Jupiler League       Belgium

   home_team_long_name  home_team_short_name  away_team_long_name  \
0             KRC Genk                   GEN         Beerschot AC
1      SV Zulte-Waregem                   ZUL     Sporting Lokeren
2     KSV Cercle Brugge                   CEB       RSC Anderlecht
3             KAA Gent                   GEN            RAEC Mons
4         FCV Dender EH                   DEN   Standard de LiÃ¨ge

   away_team_short_name
0                   BAC
1                   LOK
2                   AND
3                   MON
4                   STL

[5 rows x 121 columns]
```

```
MCLT_Combined
```

```
          id  country_id  league_id     season  stage
date  \
```

```
0          1           1           1  2008/2009      1  2008-08-17
00:00:00
1          2           1           1  2008/2009      1  2008-08-16
00:00:00
2          3           1           1  2008/2009      1  2008-08-16
00:00:00
3          4           1           1  2008/2009      1  2008-08-17
00:00:00
4          5           1           1  2008/2009      1  2008-08-16
00:00:00
...      ...         ...         ...        ...    ...
...
25974  25975       24558       24558  2015/2016      9  2015-09-22
00:00:00
25975  25976       24558       24558  2015/2016      9  2015-09-23
00:00:00
25976  25977       24558       24558  2015/2016      9  2015-09-23
00:00:00
25977  25978       24558       24558  2015/2016      9  2015-09-22
00:00:00
25978  25979       24558       24558  2015/2016      9  2015-09-23
00:00:00

       match_api_id  home_team_api_id  away_team_api_id
home_team_goal  ...  \
0            492473              9987              9993
1  ...
1            492474             10000              9994
0  ...
2            492475              9984              8635
0  ...
3            492476              9991              9998
5  ...
4            492477              7947              9985
1  ...
...             ...               ...               ...              ..
.  ...
25974       1992091             10190             10191
1  ...
25975       1992092              9824             10199
1  ...
25976       1992093              9956             10179
2  ...
25977       1992094              7896             10243
0  ...
25978       1992095             10192              9931
4  ...

        GBA   BSH   BSD   BSA                 name_league  name_country
```

```
\
0      4.00  1.73  3.40  4.20    Belgium Jupiler League       Belgium

1      3.75  1.91  3.25  3.60    Belgium Jupiler League       Belgium

2      2.50  2.30  3.20  2.75    Belgium Jupiler League       Belgium

3      5.50  1.44  3.75  6.50    Belgium Jupiler League       Belgium

4      1.65  4.75  3.30  1.67    Belgium Jupiler League       Belgium

...     ...   ...   ...   ...                       ...           ...

25974   NaN   NaN   NaN   NaN  Switzerland Super League   Switzerland

25975   NaN   NaN   NaN   NaN  Switzerland Super League   Switzerland

25976   NaN   NaN   NaN   NaN  Switzerland Super League   Switzerland

25977   NaN   NaN   NaN   NaN  Switzerland Super League   Switzerland

25978   NaN   NaN   NaN   NaN  Switzerland Super League   Switzerland


            home_team_long_name  home_team_short_name
away_team_long_name  \
0                      KRC Genk                   GEN
Beerschot AC
1                SV Zulte-Waregem                   ZUL     Sporting
Lokeren
2               KSV Cercle Brugge                   CEB       RSC
Anderlecht
3                      KAA Gent                   GEN           RAEC
Mons
4                  FCV Dender EH                   DEN   Standard de
LiÃ¨ge
...                         ...                   ...
...
25974              FC St. Gallen                   GAL              FC
Thun
25975                   FC Vaduz                   VAD            FC
Luzern
25976  Grasshopper Club ZÃ¼rich                   GRA              FC
Sion
25977                    Lugano                   LUG            FC
ZÃ¼rich
25978            BSC Young Boys                    YB             FC
Basel

       away_team_short_name
```

```
0                        BAC
1                        LOK
2                        AND
3                        MON
4                        STL
...                      ...
25974                    THU
25975                    LUZ
25976                    SIO
25977                    ZUR
25978                    BAS

[25979 rows x 121 columns]
```

Missing Data Imputation

Players Table

```
Player_Attributes_with_Names.isnull().sum()
```

```
name                                 0
full_name                            0
birth_date                           0
age                                  0
height_cm                            0
weight_kgs                           0
positions                            0
nationality                          0
overall_rating                       0
potential                            0
value_euro                         255
wage_euro                          246
preferred_foot                       0
international_reputation(1-5)        0
weak_foot(1-5)                       0
skill_moves(1-5)                     0
body_type                            0
release_clause_euro               1837
national_team                    17097
national_rating                  17097
national_team_position           17097
national_jersey_number           17097
crossing                             0
finishing                            0
heading_accuracy                     0
short_passing                        0
volleys                              0
dribbling                            0
curve                                0
freekick_accuracy                    0
```

```
long_passing                                    0
ball_control                                    0
acceleration                                    0
sprint_speed                                    0
agility                                         0
reactions                                       0
balance                                         0
shot_power                                      0
jumping                                         0
stamina                                         0
strength                                        0
long_shots                                      0
aggression                                      0
interceptions                                   0
positioning                                     0
vision                                          0
penalties                                       0
composure                                       0
marking                                         0
standing_tackle                                 0
sliding_tackle                                  0
dtype: int64
```

```
# Removing columns with more than 1000 null values
Player_Attributes_with_Names =
Player_Attributes_with_Names.drop(columns=[col for col in
Player_Attributes_with_Names.columns if
Player_Attributes_with_Names[col].isnull().sum() > 1000])

# Replacing null values in 'value_euro' and 'wage_euro' with their
averages
Player_Attributes_with_Names['value_euro'] =
Player_Attributes_with_Names['value_euro'].fillna(Player_Attributes_wi
th_Names['value_euro'].mean())
Player_Attributes_with_Names['wage_euro'] =
Player_Attributes_with_Names['wage_euro'].fillna(Player_Attributes_wit
h_Names['wage_euro'].mean())

# Checking again for null values to confirm
print(Player_Attributes_with_Names.isnull().sum())
```

```
name                                    0
full_name                               0
birth_date                              0
age                                     0
height_cm                               0
weight_kgs                              0
positions                               0
nationality                             0
overall_rating                          0
```

```
potential                            0
value_euro                           0
wage_euro                            0
preferred_foot                       0
international_reputation(1-5)        0
weak_foot(1-5)                       0
skill_moves(1-5)                     0
body_type                            0
crossing                             0
finishing                            0
heading_accuracy                     0
short_passing                        0
volleys                              0
dribbling                            0
curve                                0
freekick_accuracy                    0
long_passing                         0
ball_control                         0
acceleration                         0
sprint_speed                         0
agility                              0
reactions                            0
balance                              0
shot_power                           0
jumping                              0
stamina                              0
strength                             0
long_shots                           0
aggression                           0
interceptions                        0
positioning                          0
vision                               0
penalties                            0
composure                            0
marking                              0
standing_tackle                      0
sliding_tackle                       0
dtype: int64
```

Team Attributes Table

```
Team_Attributes_with_Names.isnull().sum()
```

```
id_x                                    0
team_api_id                             0
team_fifa_api_id_x                      0
team_long_name                          0
team_short_name                         0
id_y                                    0
team_fifa_api_id_y                      0
```

```
date                              0
buildUpPlaySpeed                  0
buildUpPlaySpeedClass             0
buildUpPlayDribbling            969
buildUpPlayDribblingClass         0
buildUpPlayPassing                0
buildUpPlayPassingClass           0
buildUpPlayPositioningClass       0
chanceCreationPassing             0
chanceCreationPassingClass        0
chanceCreationCrossing            0
chanceCreationCrossingClass       0
chanceCreationShooting            0
chanceCreationShootingClass       0
chanceCreationPositioningClass    0
defencePressure                   0
defencePressureClass              0
defenceAggression                 0
defenceAggressionClass            0
defenceTeamWidth                  0
defenceTeamWidthClass             0
defenceDefenderLineClass          0
dtype: int64

Team_Attributes_with_Names.drop(columns=['buildUpPlayDribbling'],
inplace=True)
```

Match, Country, League and Team Table

```
MCLT_Combined.isnull().sum()

id                      0
country_id              0
league_id               0
season                  0
stage                   0
                       ..
name_country            0
home_team_long_name     0
home_team_short_name    0
away_team_long_name     0
away_team_short_name    0
Length: 121, dtype: int64

# printing the null values
null_values = MCLT_Combined.isnull().sum()

# Filtering and displaying columns with no null values
no_null = null_values[null_values ==0]
print(no_null)
```

```
id                        0
country_id                0
league_id                 0
season                    0
stage                     0
date                      0
match_api_id              0
home_team_api_id          0
away_team_api_id          0
home_team_goal            0
away_team_goal            0
name_league               0
name_country              0
home_team_long_name       0
home_team_short_name      0
away_team_long_name       0
away_team_short_name      0
dtype: int64

# Listing the columns with no missing values
columns_no_null = ['id', 'country_id', 'league_id', 'season', 'stage',
'date',
                   'match_api_id', 'home_team_api_id',
'away_team_api_id',
                   'home_team_goal', 'away_team_goal', 'name_league',
'name_country',
                   'home_team_long_name','away_team_long_name']

# Creating a new DataFrame with these columns
MCLT_Combined = MCLT_Combined[columns_no_null]

# Displaying the first few rows to verify
print(MCLT_Combined.head())

   id  country_id  league_id     season  stage                 date  \
0   1           1          1  2008/2009      1  2008-08-17 00:00:00
1   2           1          1  2008/2009      1  2008-08-16 00:00:00
2   3           1          1  2008/2009      1  2008-08-16 00:00:00
3   4           1          1  2008/2009      1  2008-08-17 00:00:00
4   5           1          1  2008/2009      1  2008-08-16 00:00:00

   match_api_id  home_team_api_id  away_team_api_id  home_team_goal  \
0        492473              9987              9993               1
1        492474             10000              9994               0
2        492475              9984              8635               0
3        492476              9991              9998               5
4        492477              7947              9985               1

   away_team_goal             name_league name_country
home_team_long_name  \
```

```
0                1  Belgium Jupiler League      Belgium            KRC
Genk
1                0  Belgium Jupiler League      Belgium    SV Zulte-
Waregem
2                3  Belgium Jupiler League      Belgium   KSV Cercle
Brugge
3                0  Belgium Jupiler League      Belgium            KAA
Gent
4                3  Belgium Jupiler League      Belgium       FCV
Dender EH

  away_team_long_name
0        Beerschot AC
1    Sporting Lokeren
2      RSC Anderlecht
3           RAEC Mons
4  Standard de LiÃ¨ge
```

Feature Engineering

In this section, we perform featre engineering on our tables of interest to create new variables that will allow us to explore the data of Eurpean football with a different perspective. These feature engineered variables were used for gaining further insights through EDA and other analysis techniques.

On Players Table

```
Player_Attributes_with_Names.columns.tolist()

['name',
 'full_name',
 'birth_date',
 'age',
 'height_cm',
 'weight_kgs',
 'positions',
 'nationality',
 'overall_rating',
 'potential',
 'value_euro',
 'wage_euro',
 'preferred_foot',
 'international_reputation(1-5)',
 'weak_foot(1-5)',
 'skill_moves(1-5)',
 'body_type',
 'crossing',
 'finishing',
```

```
 'heading_accuracy',
 'short_passing',
 'volleys',
 'dribbling',
 'curve',
 'freekick_accuracy',
 'long_passing',
 'ball_control',
 'acceleration',
 'sprint_speed',
 'agility',
 'reactions',
 'balance',
 'shot_power',
 'jumping',
 'stamina',
 'strength',
 'long_shots',
 'aggression',
 'interceptions',
 'positioning',
 'vision',
 'penalties',
 'composure',
 'marking',
 'standing_tackle',
 'sliding_tackle']
```

- Age Category
- BMI
- Offensive Score
- Defensive Score

```
Player_Attributes_with_Names['age']
```

```
0        31
1        27
2        25
3        27
4        27
         ..
17949    25
17950    23
17951    22
17952    21
17953    19
Name: age, Length: 17954, dtype: int64
```

```
bins = [-1, 19, 25, 30, float('inf')]  # Use float('inf') for ages 30
and above
labels = ['Under 20', '20-25', '25-30', '30+']

# Categorizing the 'age' column
Player_Attributes_with_Names['age_category'] =
pd.cut(Player_Attributes_with_Names['age'], bins=bins, labels=labels)

# Displaying the updated DataFrame
print(Player_Attributes_with_Names[['age', 'age_category']].head())
```

```
   age age_category
0   31          30+
1   27        25-30
2   25        20-25
3   27        25-30
4   27        25-30
```

```
Player_Attributes_with_Names['height_cm']
```

```
0        170.18
1        154.94
2        190.50
3        162.56
4        187.96
          ...
17949    175.26
17950    182.88
17951    185.42
17952    175.26
17953    190.50
Name: height_cm, Length: 17954, dtype: float64
```

```
#Calculating the BMI
Player_Attributes_with_Names['height_m'] =
Player_Attributes_with_Names['height_cm'] / 100  # Convert height to
meters
Player_Attributes_with_Names['BMI'] =
Player_Attributes_with_Names['weight_kgs'] /
(Player_Attributes_with_Names['height_m'] ** 2)

# Categorizing the BMI
def categorize_bmi(bmi):
    if bmi < 18.5:
        return 'Underweight'
    elif 18.5 <= bmi < 24.9:
        return 'Normal weight'
    elif 25 <= bmi < 29.9:
        return 'Overweight'
    else:
```

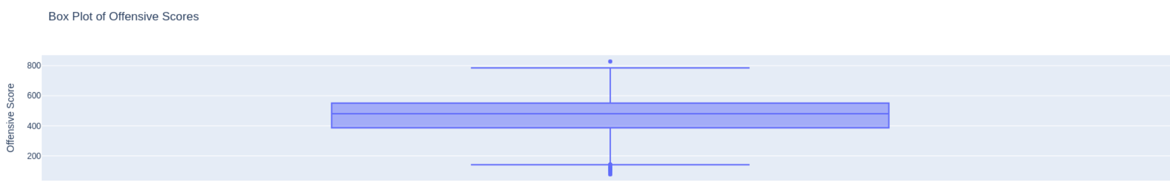
```
        return 'Obesity'

Player_Attributes_with_Names['BMI_category'] =
Player_Attributes_with_Names['BMI'].apply(categorize_bmi)

#Displaying the updated DataFrame with BMI and BMI_category
print(Player_Attributes_with_Names[['weight_kgs', 'height_cm', 'BMI',
'BMI_category']].head())
```

```
   weight_kgs  height_cm        BMI   BMI_category
0        72.1     170.18  24.895349  Normal weight
1        76.2     154.94  31.741531        Obesity
2        83.9     190.50  23.119157  Normal weight
3        59.0     162.56  22.326705  Normal weight
4        88.9     187.96  25.163491     Overweight
```

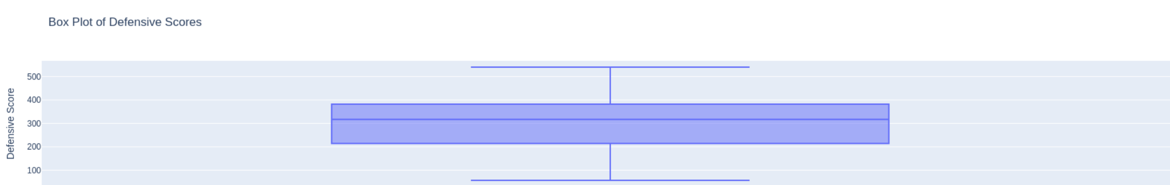
```
# Defining the offensive and defensive skill columns
offensive_skills = ['finishing', 'crossing', 'dribbling', 'volleys',
'short_passing', 'long_passing', 'ball_control', 'curve',
'freekick_accuracy']
defensive_skills = ['interceptions', 'marking', 'standing_tackle',
'sliding_tackle', 'heading_accuracy', 'aggression']

# Calculating Offensive and Defensive Scores
Player_Attributes_with_Names['offensive_score'] =
Player_Attributes_with_Names[offensive_skills].sum(axis=1)
Player_Attributes_with_Names['defensive_score'] =
Player_Attributes_with_Names[defensive_skills].sum(axis=1)

# Displaying the updated DataFrame with new scores
print(Player_Attributes_with_Names[['name', 'offensive_score',
'defensive_score']].head())
```

```
           name  offensive_score  defensive_score
0      L. Messi              828              227
1    C. Eriksen              777              292
2      P. Pogba              760              414
3    L. Insigne              747              213
4  K. Koulibaly              382              524
```

```
# Creating the box plot
offensive_box_plot = px.box(
    Player_Attributes_with_Names,
    y='offensive_score',
    title='Box Plot of Offensive Scores',
    labels={'offensive_score': 'Offensive Score'}
)


offensive_box_plot.show()
```




```
# Creating the box plot
defensive_box_plot = px.box(
    Player_Attributes_with_Names,
    y='defensive_score',
    title='Box Plot of Defensive Scores',
    labels={'defensive_score': 'Defensive Score'},
    )

defensive_box_plot.show()
```




```
# Defining bins for offensive scores
offensive_bins = [0, 200, 500, 800]
offensive_labels = ['Low', 'Medium', 'High']

# Categorizing scores into bins
Player_Attributes_with_Names['offensive_category'] = pd.cut(
    Player_Attributes_with_Names['offensive_score'],
    bins=offensive_bins,
    labels=offensive_labels,
    right=False
)

Player_Attributes_with_Names['offensive_category']

0           NaN
1          High
2          High
3          High
4        Medium
          ...
17949      High
17950    Medium
17951       Low
```

```
17952    Medium
17953    Medium
Name: offensive_category, Length: 17954, dtype: category
Categories (3, object): ['Low' < 'Medium' < 'High']
```

```
# Defining the bins for defensive scores
defensive_bins = [0, 200, 500, 800]
defensive_labels = ['Low', 'Medium', 'High']

# Categorizing the scores into bins
Player_Attributes_with_Names['defensive_category'] = pd.cut(
    Player_Attributes_with_Names['defensive_score'],
    bins=defensive_bins,
    labels=defensive_labels,
    right=False
)
```

```
Player_Attributes_with_Names['defensive_score']
```

```
0        227
1        292
2        414
3        213
4        524
        ...
17949    248
17950    347
17951     84
17952    159
17953    220
Name: defensive_score, Length: 17954, dtype: int64
```

```
Player_Attributes_with_Names.shape
```

```
(17954, 54)
```

```
Player_Attributes_with_Names['overall_rating'].describe()
```

```
count    17954.000000
mean        66.240169
std          6.963730
min         47.000000
25%         62.000000
50%         66.000000
75%         71.000000
max         94.000000
Name: overall_rating, dtype: float64
```

Performance Category Based on Overal_rating

```
def categorize_rating(overall_rating):
    if overall_rating < 65:
        return 'Low'
    elif 65 <= overall_rating < 80:
        return 'Medium'
    else:
        return 'High'

# Creating a new column for performance categories
Player_Attributes_with_Names['performance_category'] =
Player_Attributes_with_Names['overall_rating'].apply(categorize_rating
)
```

file_path = '/content/drive/MyDrive/FootBall/Final Tables/Player_Attributes_with_Names.csv'
Player_Attributes_with_Names.to_csv(file_path, index=False)

On MCLT Table

```
MCLT_Combined.columns

Index(['id', 'country_id', 'league_id', 'season', 'stage', 'date',
       'match_api_id', 'home_team_api_id', 'away_team_api_id',
       'home_team_goal', 'away_team_goal', 'name_league',
'name_country',
       'home_team_long_name', 'away_team_long_name'],
      dtype='object')
```

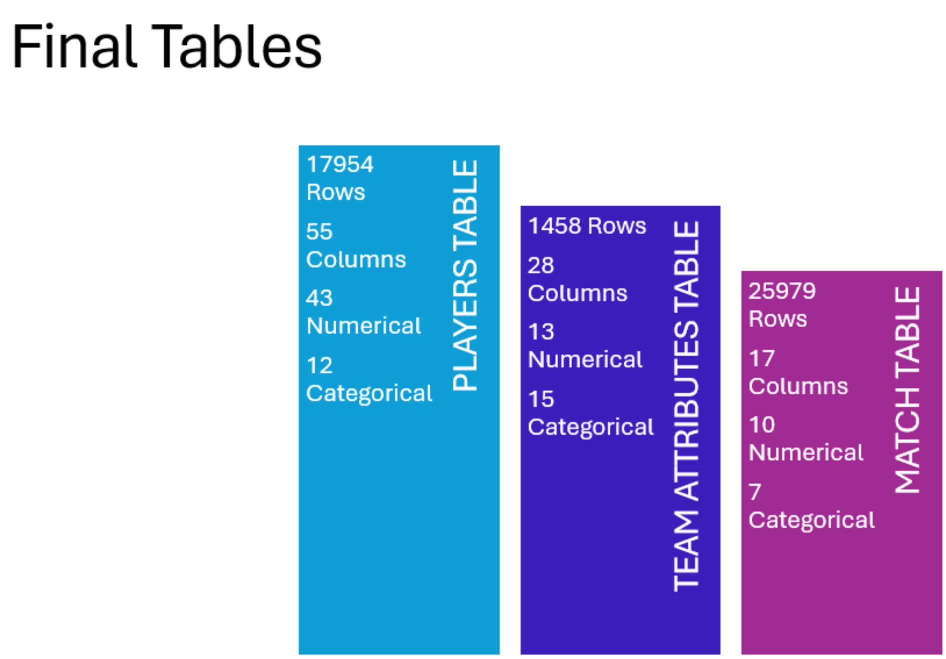
- Match Result
- Total Goals

```
MCLT_Combined['home_team_goal']

0        1
1        0
2        0
3        5
4        1
        ..
25974    1
25975    1
25976    2
25977    0
25978    4
Name: home_team_goal, Length: 25979, dtype: int64

# Adding a column for match result
def match_result(row):
    if row['home_team_goal'] > row['away_team_goal']:
        return 'Home Win'
    elif row['home_team_goal'] < row['away_team_goal']:
```

```
        return 'Away Win'
    else:
        return 'Draw'

MCLT_Combined['match_result'] = MCLT_Combined.apply(match_result, axis=1)

# Adding a new column for total goals
MCLT_Combined['total_goals'] = MCLT_Combined['home_team_goal'] + MCLT_Combined['away_team_goal']
```

Final Tables

In this section we list the final tables that we derived after performing preprocessing and transformations.



Player_Attributes_with_Names

```
Player_Attributes_with_Names.describe()
```

```
                age      height_cm     weight_kgs  overall_rating
potential  \
count  17954.000000  17954.000000  17954.000000    17954.000000
17954.000000
mean      25.565445    174.946921     75.301047       66.240169
71.430935
```

```
std         4.705708     14.029449      7.083684        6.963730
6.131339
min        17.000000    152.400000     49.900000       47.000000
48.000000
25%        22.000000    154.940000     69.900000       62.000000
67.000000
50%        25.000000    175.260000     74.800000       66.000000
71.000000
75%        29.000000    185.420000     79.800000       71.000000
75.000000
max        46.000000    205.740000    110.200000       94.000000
95.000000

         value_euro      wage_euro  international_reputation(1-5)  \
count  1.795400e+04   17954.000000                   17954.000000
mean   2.479280e+06    9902.134628                       1.111674
std    5.646481e+06   21844.377245                       0.392168
min    1.000000e+04    1000.000000                       1.000000
25%    3.250000e+05    1000.000000                       1.000000
50%    7.250000e+05    3000.000000                       1.000000
75%    2.300000e+06    9902.134628                       1.000000
max    1.105000e+08  565000.000000                       5.000000

       weak_foot(1-5)  skill_moves(1-5)  ...        vision
penalties  \
count    17954.000000      17954.000000  ...  17954.000000
17954.000000
mean         2.945695          2.361034  ...     53.406260
48.357302
std          0.663691          0.763223  ...     14.156038
15.810844
min          1.000000          1.000000  ...     10.000000
5.000000
25%          3.000000          2.000000  ...     44.000000
38.000000
50%          3.000000          2.000000  ...     55.000000
49.000000
75%          3.000000          3.000000  ...     64.000000
60.000000
max          5.000000          5.000000  ...     94.000000
92.000000

          composure       marking  standing_tackle  sliding_tackle  \
count  17954.000000  17954.000000     17954.000000    17954.000000
mean      58.680183     47.162861        47.733040       45.705915
std       11.625541     20.037346        21.674973       21.285812
min       12.000000      3.000000         2.000000        3.000000
25%       51.000000     30.000000        27.000000       24.000000
50%       60.000000     52.500000        55.000000       52.000000
75%       67.000000     64.000000        66.000000       64.000000
```

```
max        96.000000    94.000000        93.000000       90.000000

           height_m           BMI  offensive_score  defensive_score
count  17954.000000  17954.000000     17954.000000     17954.000000
mean       1.749469     24.933142       452.336749       295.224518
std        0.140294      3.868855       141.905066       107.192259
min        1.524000     16.954928        77.000000        57.000000
25%        1.549400     22.365014       387.000000       214.000000
50%        1.752600     23.473057       480.000000       317.000000
75%        1.854200     27.034453       551.000000       382.000000
max        2.057400     47.447317       828.000000       540.000000

[8 rows x 43 columns]
```

```
Player_Attributes_with_Names.dtypes
```

```
name                                object
full_name                           object
birth_date                          object
age                                  int64
height_cm                          float64
weight_kgs                         float64
positions                           object
nationality                         object
overall_rating                       int64
potential                            int64
value_euro                         float64
wage_euro                          float64
preferred_foot                      object
international_reputation(1-5)        int64
weak_foot(1-5)                       int64
skill_moves(1-5)                     int64
body_type                           object
crossing                             int64
finishing                            int64
heading_accuracy                     int64
short_passing                        int64
volleys                              int64
dribbling                            int64
curve                                int64
freekick_accuracy                    int64
long_passing                         int64
ball_control                         int64
acceleration                         int64
sprint_speed                         int64
agility                              int64
reactions                            int64
balance                              int64
shot_power                           int64
jumping                              int64
```

```
stamina                                    int64
strength                                   int64
long_shots                                 int64
aggression                                 int64
interceptions                              int64
positioning                                int64
vision                                     int64
penalties                                  int64
composure                                  int64
marking                                    int64
standing_tackle                            int64
sliding_tackle                             int64
age_category                            category
height_m                                 float64
BMI                                      float64
BMI_category                              object
offensive_score                            int64
defensive_score                            int64
offensive_category                      category
defensive_category                      category
performance_category                      object
dtype: object
```

```
Player_Attributes_with_Names.shape
```

```
(17954, 55)
```

```
# Getting the data types of all columns
data_types = Player_Attributes_with_Names.dtypes

# Counting numerical columns
num_cols = sum(data_types == 'int64') + sum(data_types == 'float64')

# Counting categorical columns
cat_cols = len(data_types) - num_cols

print("Number of numerical columns:", num_cols)
print("Number of categorical columns:", cat_cols)
```

```
Number of numerical columns: 43
Number of categorical columns: 12
```

Team_Attributes_with_Names

```
Team_Attributes_with_Names.describe()
```

```
               id_x   team_api_id  team_fifa_api_id_x         id_y  \
count   1458.000000   1458.000000         1458.000000  1458.000000
mean   22692.858711   9995.727023        17706.982167   729.500000
std    15015.159107  13264.869900        39179.857739   421.032659
```

```
min        1.000000    1601.000000           1.000000     1.000000
25%     9547.250000    8457.750000         110.000000   365.250000
50%    20524.500000    8674.000000         485.000000   729.500000
75%    35294.000000    9904.000000        1900.000000  1093.750000
max    50204.000000  274581.000000      112513.000000  1458.000000

       team_fifa_api_id_y  buildUpPlaySpeed  buildUpPlayPassing  \
count         1458.000000       1458.000000         1458.000000
mean         17706.982167         52.462277           48.490398
std          39179.857739         11.545869           10.896101
min              1.000000         20.000000           20.000000
25%            110.000000         45.000000           40.000000
50%            485.000000         52.000000           50.000000
75%           1900.000000         62.000000           55.000000
max         112513.000000         80.000000           80.000000

       chanceCreationPassing  chanceCreationCrossing
chanceCreationShooting  \
count            1458.000000             1458.000000
1458.000000
mean               52.165295               53.731824
53.969136
std                10.360793               11.086796
10.327566
min                21.000000               20.000000
22.000000
25%                46.000000               47.000000
48.000000
50%                52.000000               53.000000
53.000000
75%                59.000000               62.000000
61.000000
max                80.000000               80.000000
80.000000

       defencePressure  defenceAggression  defenceTeamWidth
count      1458.000000        1458.000000       1458.000000
mean         46.017147          49.251029         52.185871
std          10.227225           9.738028          9.574712
min          23.000000          24.000000         29.000000
25%          39.000000          44.000000         47.000000
50%          45.000000          48.000000         52.000000
75%          51.000000          55.000000         58.000000
max          72.000000          72.000000         73.000000
```

```
Team_Attributes_with_Names.dtypes
```

```
id_x                                int64
team_api_id                         int64
team_fifa_api_id_x                float64
```

```
team_long_name                      object
team_short_name                     object
id_y                                 int64
team_fifa_api_id_y                   int64
date                                object
buildUpPlaySpeed                     int64
buildUpPlaySpeedClass               object
buildUpPlayDribblingClass           object
buildUpPlayPassing                   int64
buildUpPlayPassingClass             object
buildUpPlayPositioningClass         object
chanceCreationPassing                int64
chanceCreationPassingClass          object
chanceCreationCrossing               int64
chanceCreationCrossingClass         object
chanceCreationShooting               int64
chanceCreationShootingClass         object
chanceCreationPositioningClass      object
defencePressure                      int64
defencePressureClass                object
defenceAggression                    int64
defenceAggressionClass              object
defenceTeamWidth                     int64
defenceTeamWidthClass               object
defenceDefenderLineClass            object
dtype: object
```

```
Team_Attributes_with_Names.shape
```

```
(1458, 28)
```

```
# Getting the data types of all columns
data_types = Team_Attributes_with_Names.dtypes

# Counting the numerical columns
num_cols = sum(data_types == 'int64') + sum(data_types == 'float64')

# Counting categorical columns
cat_cols = len(data_types) - num_cols

print("Number of numerical columns:", num_cols)
print("Number of categorical columns:", cat_cols)
```

```
Number of numerical columns: 13
Number of categorical columns: 15
```

Match ,Country ,League Table

```
MCLT_Combined.describe()
```

```
                 id    country_id     league_id         stage
match_api_id  \
count  25979.000000  25979.000000  25979.000000  25979.000000
2.597900e+04
mean   12990.000000  11738.630317  11738.630317     18.242773
1.195429e+06
std     7499.635658   7553.936759   7553.936759     10.407354
4.946279e+05
min        1.000000      1.000000      1.000000      1.000000
4.831290e+05
25%     6495.500000   4769.000000   4769.000000      9.000000
7.684365e+05
50%    12990.000000  10257.000000  10257.000000     18.000000
1.147511e+06
75%    19484.500000  17642.000000  17642.000000     27.000000
1.709852e+06
max    25979.000000  24558.000000  24558.000000     38.000000
2.216672e+06

       home_team_api_id  away_team_api_id  home_team_goal
away_team_goal  \
count      25979.000000      25979.000000    25979.000000
25979.000000
mean        9984.371993       9984.475115        1.544594
1.160938
std        14087.453758      14087.445135        1.297158
1.142110
min         1601.000000       1601.000000        0.000000
0.000000
25%         8475.000000       8475.000000        1.000000
0.000000
50%         8697.000000       8697.000000        1.000000
1.000000
75%         9925.000000       9925.000000        2.000000
2.000000
max       274581.000000     274581.000000       10.000000
9.000000

        total_goals
count  25979.000000
mean       2.705531
std        1.672456
min        0.000000
25%        2.000000
50%        3.000000
75%        4.000000
max       12.000000

MCLT_Combined.dtypes
```

```
id                          int64
country_id                  int64
league_id                   int64
season                     object
stage                       int64
date                       object
match_api_id                int64
home_team_api_id            int64
away_team_api_id            int64
home_team_goal              int64
away_team_goal              int64
name_league                object
name_country               object
home_team_long_name        object
away_team_long_name        object
match_result               object
total_goals                 int64
dtype: object
```

```
MCLT_Combined.shape
```

```
(25979, 17)
```

```
# Getting the data types of all columns
data_types = MCLT_Combined.dtypes

# Counting the numerical columns
num_cols = sum(data_types == 'int64') + sum(data_types == 'float64')

# Counting the categorical columns
cat_cols = len(data_types) - num_cols

print("Number of numerical columns:", num_cols)
print("Number of categorical columns:", cat_cols)
```

```
Number of numerical columns: 10
Number of categorical columns: 7
```

Exploratory Data Analysis

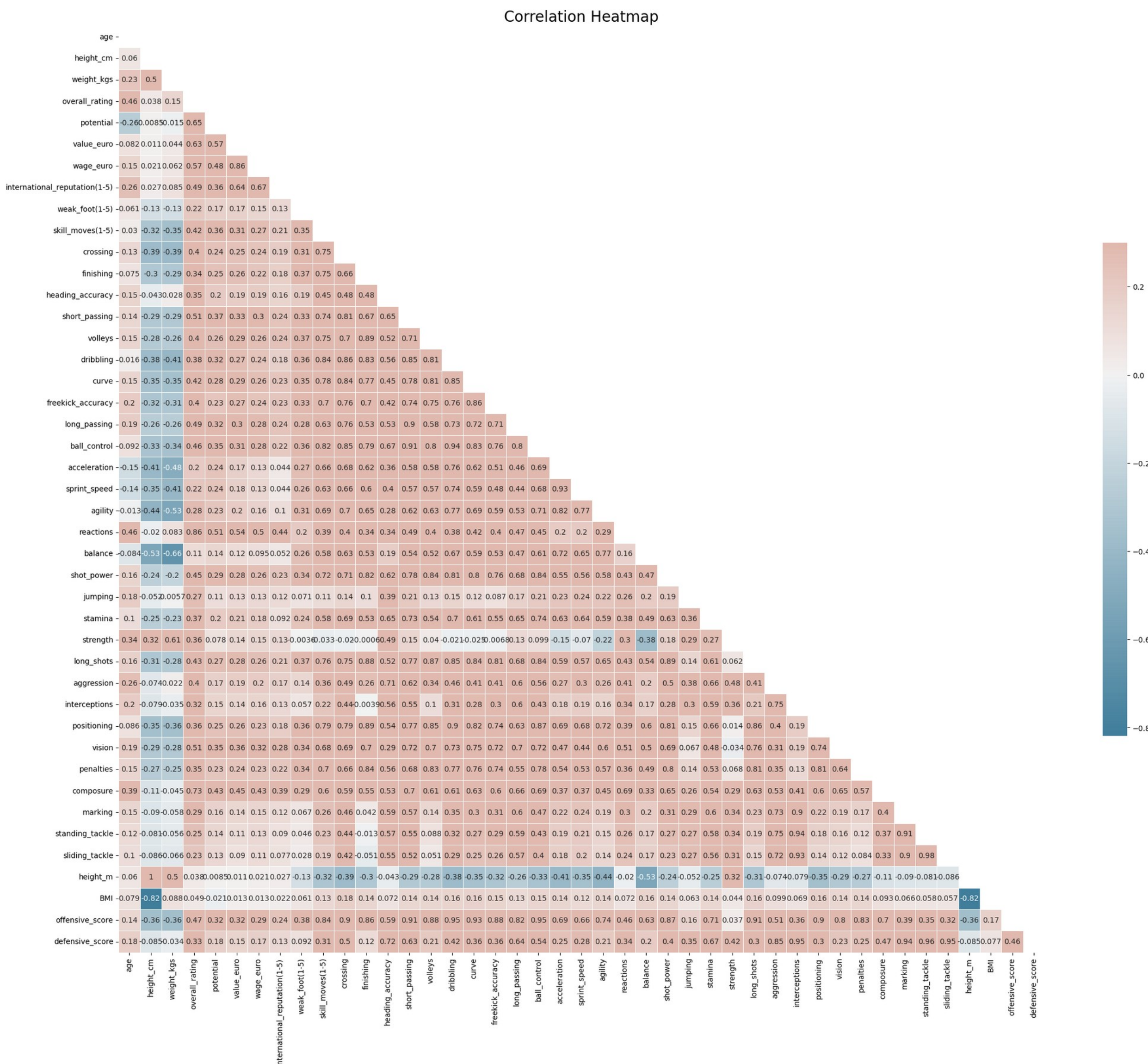
Exploratory Data Analysis (EDA) is the initial phase of data analysis where we explore the dataset to uncover patterns, relationships, and anomalies. In this project, we performed EDA to gain a deeper understanding of player, match and team attributes by plotting various visualizations such as correlation heatmaps, distribution plots, and scatter plots. These graphs helped us identify significant relationships between variables.

Correlations

In this section, we analyze the correlations between various attributes of both players and teams to uncover significant relationships within the dataset. Using correlation matrices and heatmaps, we explore how different player skills, team strategies, and other features influence one another. By examining these correlations, we can identify patterns, such as how certain skills are correlatd. This analysis provides a deeper understanding of our dataset

The following correlation heatmap visualizes the relationships between different player attributes in the dataset. The colors represent the strength of the correlation, with red indicating a positive correlation and blue representing a negative correlation.

```
# Selecting only numerical columns
numerical_df =
Player_Attributes_with_Names.select_dtypes(include=[np.number])

# Computing the correlation matrix
corr = numerical_df.corr()

# Generating a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Setting up the matplotlib figure
plt.figure(figsize=(28, 24))

# Generating a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Drawing the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
annot=True)

# Setting up the title
plt.title('Correlation Heatmap', fontsize=20)

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



High Positive Correlations:

- Attributes such as short_passing, ball_control, and dribbling are strongly correlated with each other (values close to 0.8), suggesting that players who are good at one of these skills tend to excel at the others.
- strength and heading_accuracy (0.62) show a strong positive relationship, indicating that stronger players tend to perform better in heading.
- offensive_score shows strong positive correlations with attributes like finishing (0.76) and positioning (0.72), indicating that offensive-minded players tend to excel in scoring-related metrics.

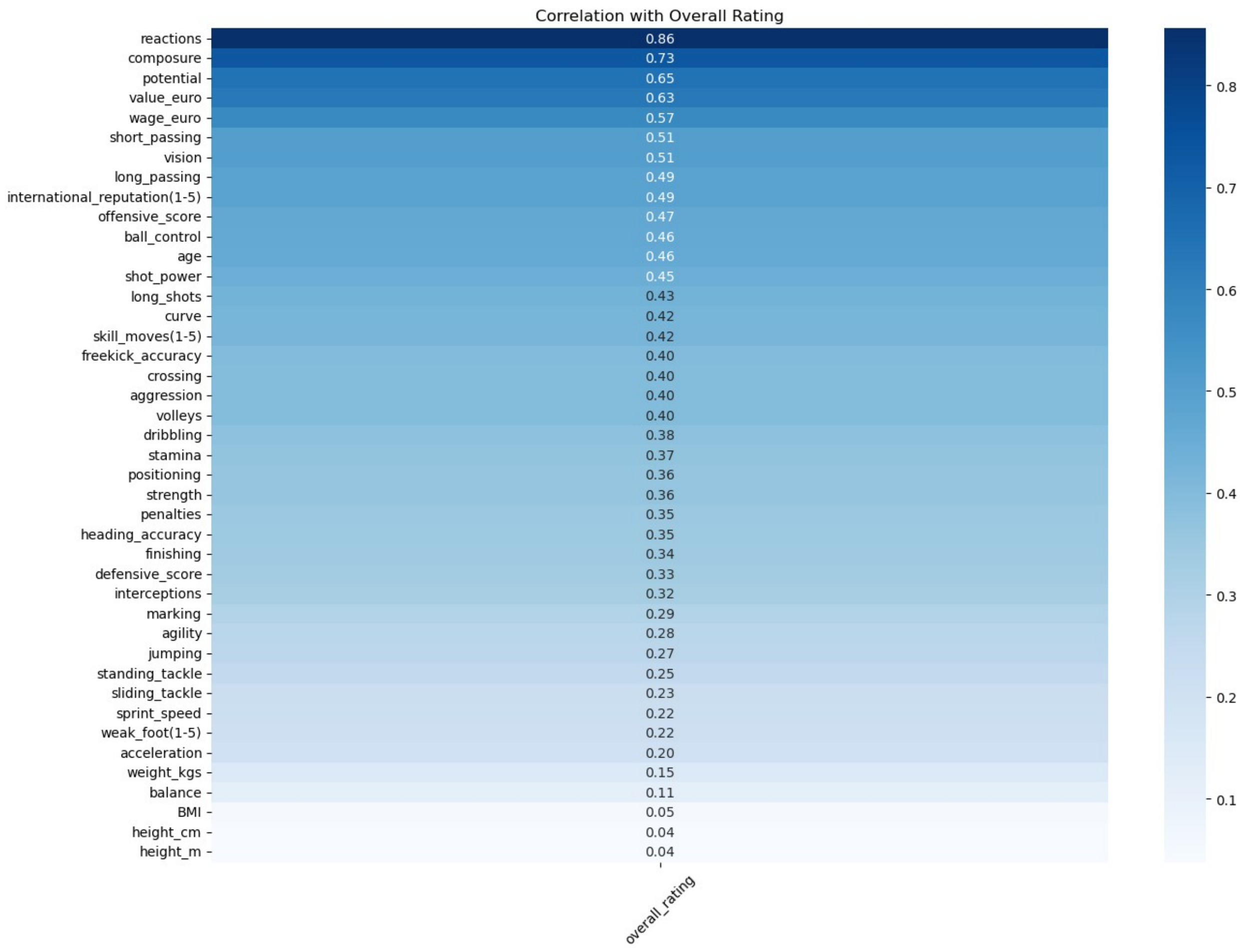
Low or Negative Correlations:

- Some attributes like balance and strength have a negative correlation (-0.45), suggesting that players with higher strength tend to have lower balance.

- defensive_score is negatively correlated with attributes like dribbling and ball_control, indicating that players who excel in defense typically have lower values for offensive skills.

Below we plot a heatmap that shows the correlation between various player attributes and the overall rating. Darker blue shades indicate higher positive correlations with the player's overall rating, while lighter shades indicate weaker correlations.

```
# Filtering the DataFrame to include only numerical columns
numeric_data =
Player_Attributes_with_Names.select_dtypes(include=['int64',
'float64'])

# Computing the correlation matrix
corr = numeric_data.corr()

# Extracting correlations with the target variable 'overall_rating'
target_variable = 'overall_rating'
target_corr = corr[target_variable].drop(target_variable)

# Sorting the correlation values in descending order
target_corr_sorted = target_corr.sort_values(ascending=False)

# Creating a heatmap for the correlation with 'overall_rating'
plt.figure(figsize=(14, 10))
sns.heatmap(target_corr_sorted.to_frame(), cmap="Blues", annot=True,
fmt='.2f', cbar=True)
plt.title('Correlation with Overall Rating')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Strong Correlations:

- Reactions (0.86), composure (0.73), and potential (0.65) show the strongest correlations with a player's overall rating. This suggests that players with higher reaction speeds and composure tend to have higher overall ratings.
- Monetary values like value_euro (0.63) and wage_euro (0.57) are also highly correlated with overall rating, meaning more highly rated players tend to be more valuable and earn higher wages.

Moderate Correlations:

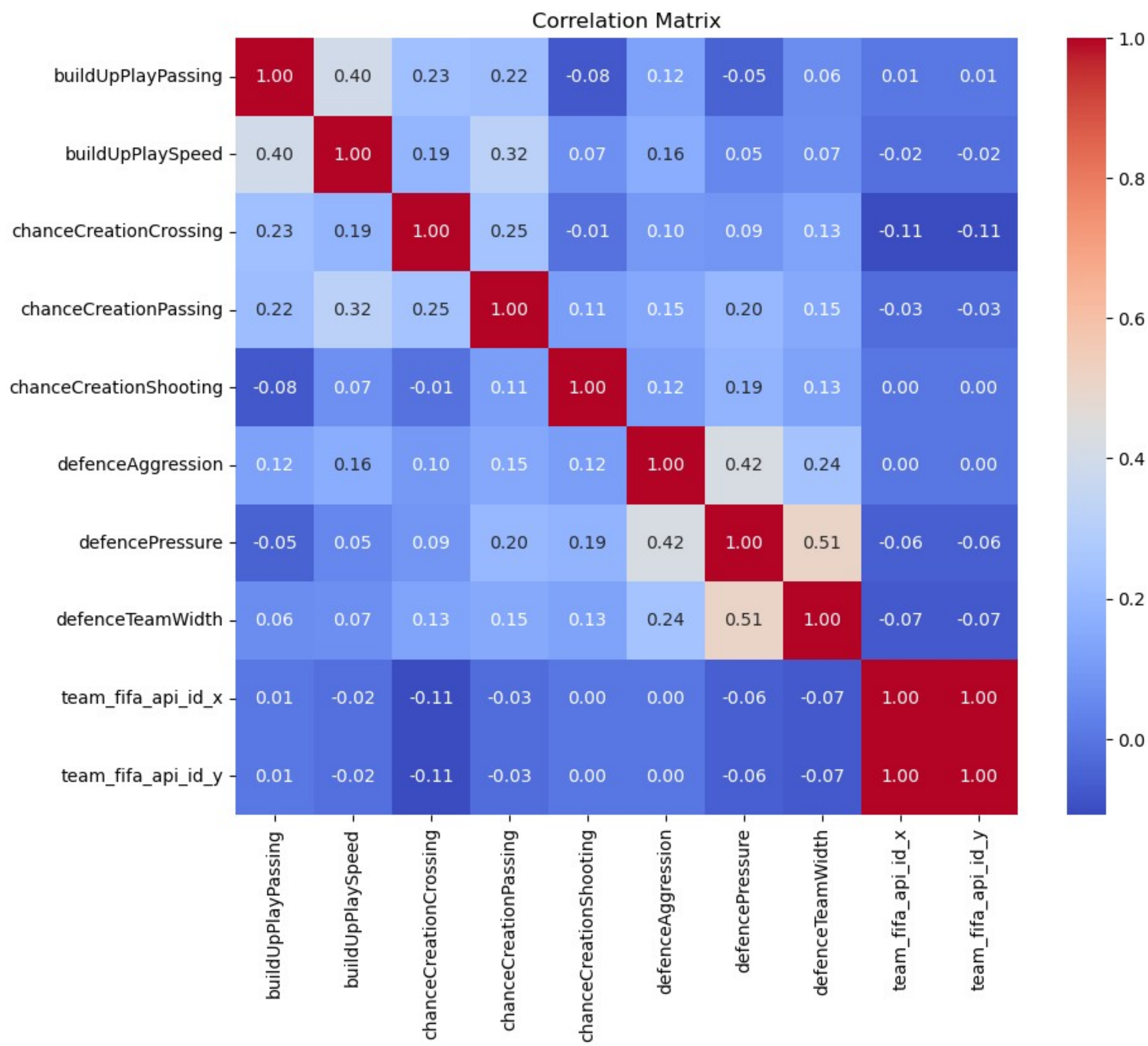
- Skills like short_passing (0.51), vision (0.51), and long_passing (0.49) are moderately correlated with overall rating, indicating that better passers are generally rated higher.
- Other attributes like offensive_score (0.47) and ball_control (0.46) also have a notable relationship with overall rating, suggesting these contribute significantly to a player's perceived value.

Lower Correlations:

- Height (0.04), weight (0.15), and BMI (0.05) show very weak correlations, indicating that physical attributes like these have little to no impact on a player's overall rating.

- Defensive metrics like standing_tackle (0.25) and interceptions (0.32) are less strongly correlated with overall rating compared to offensive or technical skills.

```
# Removing unwanted columns before calculating correlation
columns_to_remove = ['id_x', 'id_y',
'team_api_id','team_fifa_api_id','fuzzy_cluster']

# Selecting the numerical columns excluding the unwanted ones
numerical_columns =
Team_Attributes_with_Names.select_dtypes(include=['float64',
'int64']).columns
numerical_columns = numerical_columns.difference(columns_to_remove)

# Calculating the correlation matrix
correlation_matrix =
Team_Attributes_with_Names[numerical_columns].corr()

# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



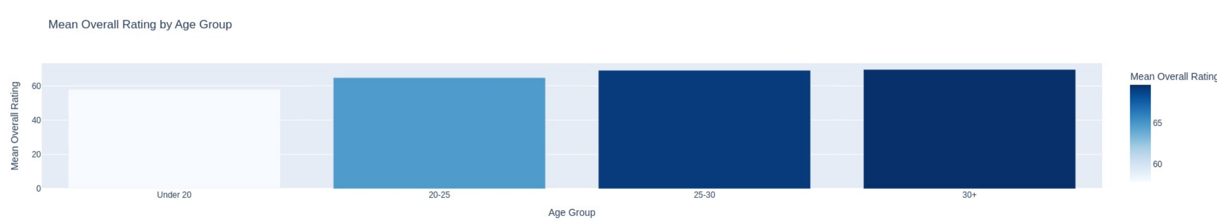
This correlation matrix shows the relationships between various team attributes such as build-up play, chance creation, and defense strategies. Notable insights include a moderate positive correlation between buildUpPlayPassing and buildUpPlaySpeed (0.40), suggesting that teams with faster build-up play tend to focus on passing strategies. There is also a strong relationship between defencePressure and defenceAggression (0.42), indicating that more aggressive teams often apply higher defensive pressure. DefencePressure and defenceTeamWidth are also positively correlated (0.51), suggesting that teams applying high pressure often have wider defensive formations. Overall, this matrix helps us understand how different tactical aspects are related in team play.

Univariate Analysis

Bar Plots

What age group of players have high overall rating?

```
# Grouping by 'age_category' and calculating the mean overall rating
mean_overall_rating =
Player_Attributes_with_Names.groupby('age_category')
['overall_rating'].mean()

# Identifying the age group with the highest overall rating
highest_rating_group = mean_overall_rating.idxmax()
highest_rating_value = mean_overall_rating.max()

# Printing the results
print(f"The age group with the highest overall rating is
'{highest_rating_group}' with an average rating of
{highest_rating_value:.2f}.")

# Converting the Series to a DataFrame for plotting
mean_rating_df = mean_overall_rating.reset_index()

# Creating a Plotly bar plot
fig = px.bar(mean_rating_df, x='age_category', y='overall_rating',
             title='Mean Overall Rating by Age Group',
             labels={'overall_rating': 'Mean Overall Rating',
'age_category': 'Age Group'},
             color='overall_rating',
             color_continuous_scale=px.colors.sequential.Blues)


fig.show()

The age group with the highest overall rating is '30+' with an average
rating of 69.51.
```

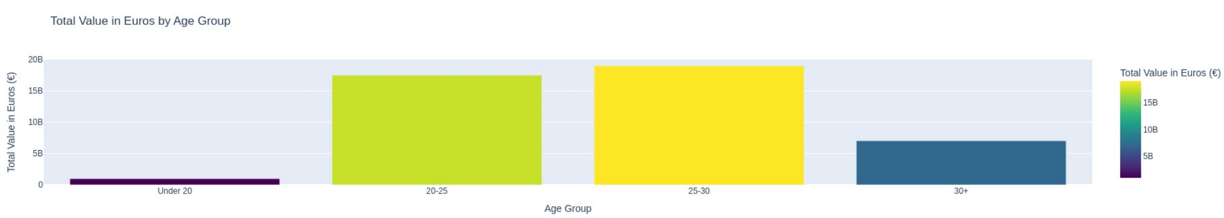


What age group of players have highest 'value euro' (overall sum)?

```
# Grouping by 'age_category' and calculating the sum of 'value_euro'
total_value_euro =
Player_Attributes_with_Names.groupby('age_category')
['value_euro'].sum()

#  Converting the series to a DataFrame for easier plotting
total_value_df = total_value_euro.reset_index()

# Identifying the age group with the highest total value in euros
```

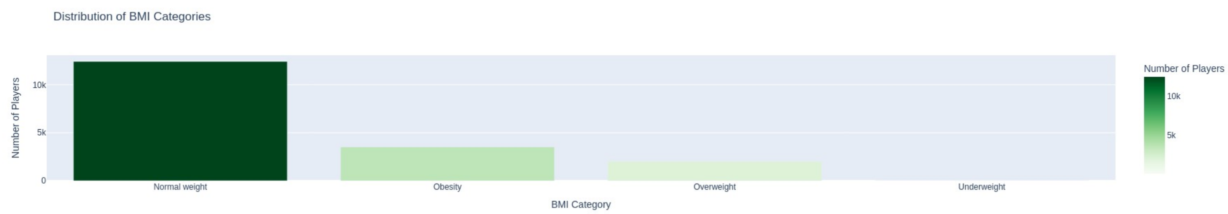
```
highest_value_group = total_value_euro.idxmax()
highest_value_sum = total_value_euro.max()

# Printing the results
print(f"The age group with the highest total value in euros is
'{highest_value_group}' with a total value of €
{highest_value_sum:.2f}.")

# Creating a Plotly bar plot
fig = px.bar(total_value_df, x='age_category', y='value_euro',
             title='Total Value in Euros by Age Group',
             labels={'value_euro': 'Total Value in Euros (€)',
'age_category': 'Age Group'},
             color='value_euro',
             color_continuous_scale=px.colors.sequential.Viridis)


fig.show()
```

```
The age group with the highest total value in euros is '25-30' with a
total value of €19010682979.83.
```



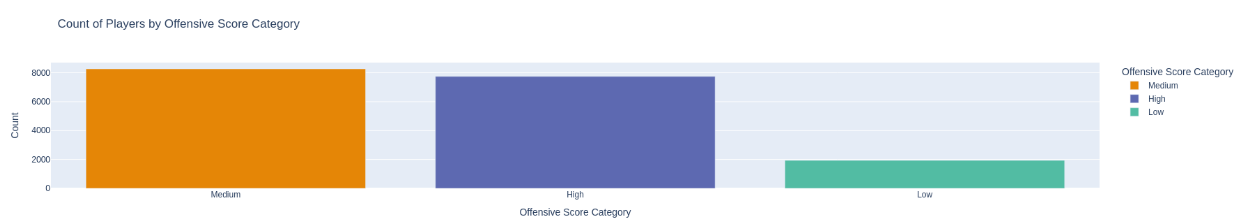
What BMI categories the players fall into?

```
bmi_counts =
Player_Attributes_with_Names['BMI_category'].value_counts().reset_inde
x()
bmi_counts.columns = ['BMI_category', 'Count']

# Creating a bar plot using Plotly
fig = px.bar(bmi_counts, x='BMI_category', y='Count',
             title='Distribution of BMI Categories',
             labels={'Count': 'Number of Players', 'BMI_category':
'BMI Category'},
             color='Count',
             color_continuous_scale=px.colors.sequential.Greens)

fig.show()
```



How many players have high Offensive Score Category

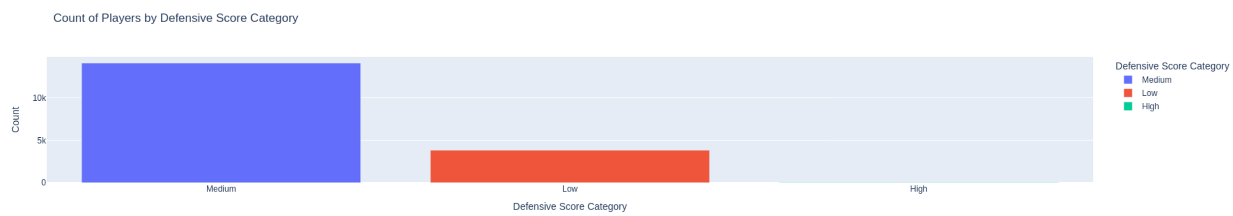
```
# Counting the occurrences in each offensive category
offensive_counts =
Player_Attributes_with_Names['offensive_category'].value_counts().rese
t_index()
offensive_counts.columns = ['offensive_category', 'count']

offensive_fig = px.bar(
    offensive_counts,
    x='offensive_category',
    y='count',
    labels={'offensive_category': 'Offensive Score Category', 'count':
'Count'},
    title='Count of Players by Offensive Score Category',
    color='offensive_category',
    color_discrete_sequence=px.colors.qualitative.Vivid
)


offensive_fig.show()
```



What category of defensive score most players fall into?

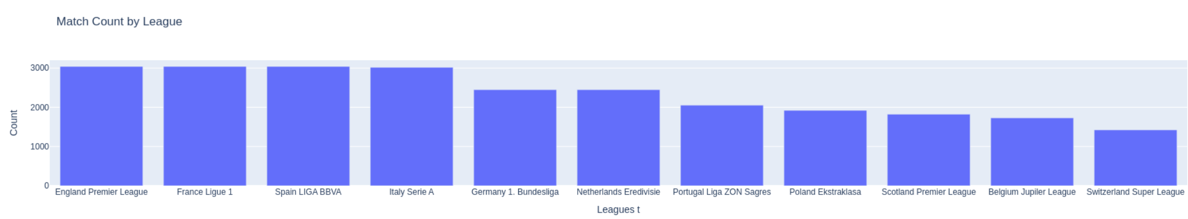
```
defensive_counts =
Player_Attributes_with_Names['defensive_category'].value_counts().rese
t_index()
defensive_counts.columns = ['defensive_category', 'count']

defensive_fig = px.bar(
    defensive_counts,
    x='defensive_category',
    y='count',
```

```
    labels={'defensive_category': 'Defensive Score Category', 'count': 
'Count'},
    title='Count of Players by Defensive Score Category',
    color='defensive_category',
    color_discrete_sequence=px.colors.qualitative.Plotly
)


defensive_fig.show()
```



How many matched were played per each league?

```
# Bar chart of match results
match_result_counts = 
MCLT_Combined['name_league'].value_counts().reset_index()
match_result_counts.columns = ['name_league', 'count']

fig_match_result = px.bar(match_result_counts,
                          x='name_league', y='count',
                          title='Match Count by League',
                          labels={'name_league': 'Leagues t', 'count': 
'Count'})
fig_match_result.show()
```



What is the most frequent match outcome (Home Win, Away Win, or Draw) based on the match result distribution?

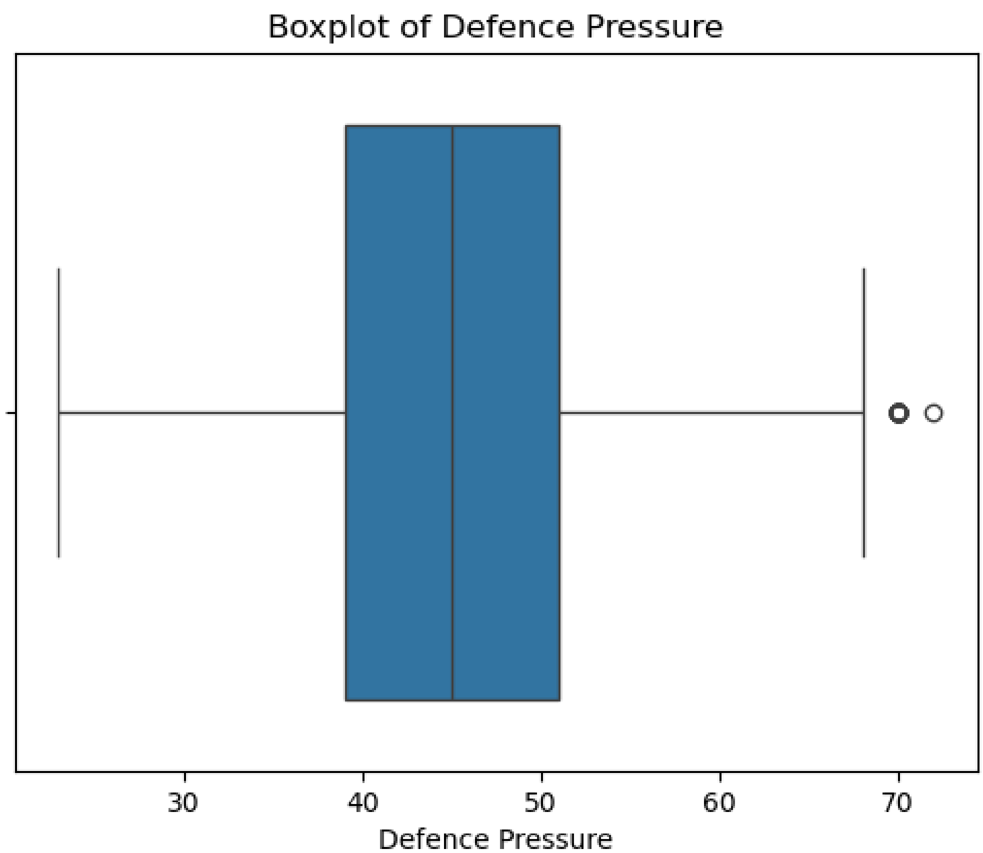
```
# Bar chart of match results
match_result_counts = 
MCLT_Combined['match_result'].value_counts().reset_index()
match_result_counts.columns = ['match_result', 'count']

fig_match_result = px.bar(match_result_counts,
```

```
                          x='match_result', y='count',
                          title='Match Result Distribution',
                          labels={'match_result': 'Match Result',
'count': 'Count'})
fig_match_result.show()
```

Box Plots

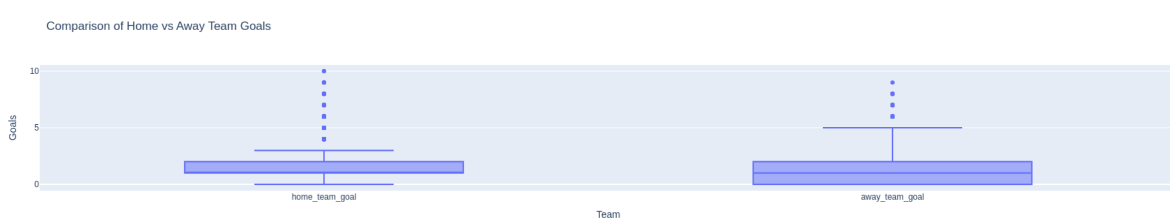
What is the typical range of "Defence Pressure" values?

```
# Boxplot for defencePressure
sns.boxplot(x=Team_Attributes_with_Names['defencePressure'])
plt.title('Boxplot of Defence Pressure')
plt.xlabel('Defence Pressure')
plt.show()
```



What is the typical goal distribution for home and away teams?

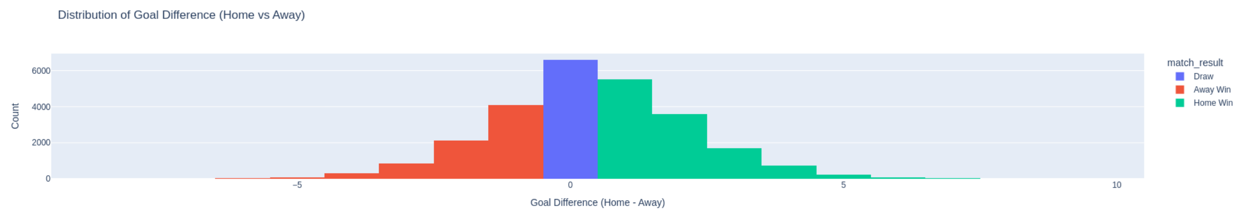
```
# Box plot comparing home and away team goals
fig_goals_comparison = px.box(MCLT_Combined.melt(id_vars=['season'],
value_vars=['home_team_goal', 'away_team_goal']),
                              x='variable', y='value',
title='Comparison of Home vs Away Team Goals', labels={'value':
'Goals', 'variable': 'Team'})
fig_goals_comparison.show()
```



Distributions

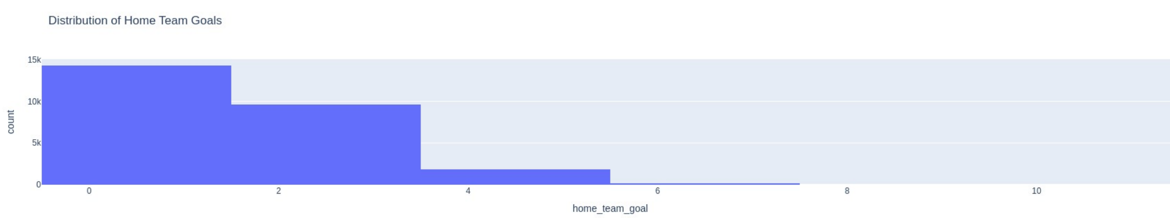
How does the goal difference vary between home and away matches?

```
# Adding a new column
MCLT_Combined['goal_difference'] = MCLT_Combined['home_team_goal'] -
MCLT_Combined['away_team_goal']

# Histogram of goal difference
fig_goal_diff = px.histogram(MCLT_Combined, x='goal_difference',
nbins=20, color='match_result',
                             title='Distribution of Goal Difference
(Home vs Away)')
fig_goal_diff.update_layout(xaxis_title='Goal Difference (Home -
Away)', yaxis_title='Count')
fig_goal_diff.show()
```



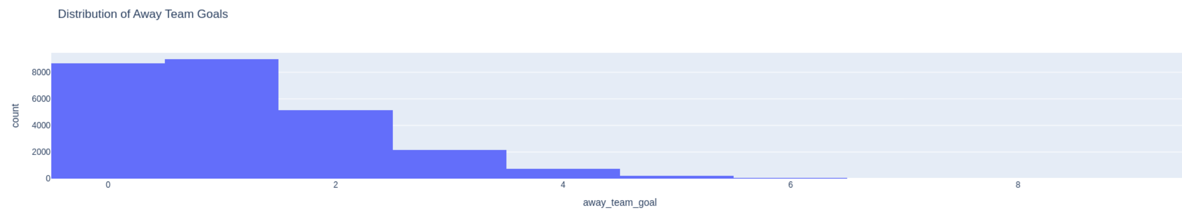
How are the goals distributed by home teams in matches?

```
# Histogram of home team goals
fig_home_goals = px.histogram(MCLT_Combined, x='home_team_goal',
nbins=10, title='Distribution of Home Team Goals')
fig_home_goals.show()
```



How are the goals distributed by away teams in matches?

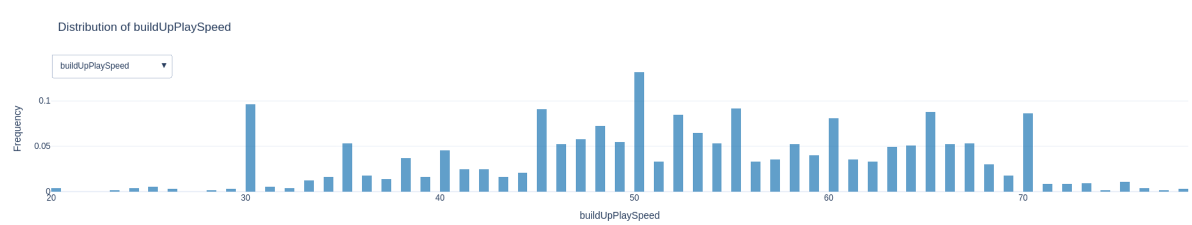
```
# Histogram of away team goals
fig_away_goals = px.histogram(MCLT_Combined, x='away_team_goal',
nbins=10, title='Distribution of Away Team Goals')
fig_away_goals.show()
```



Interactive Distribution of Team Performance Metrics

```
team_selected_columns = [
 'buildUpPlaySpeed', 'buildUpPlaySpeedClass',
       'buildUpPlayDribblingClass', 'buildUpPlayPassing',
       'buildUpPlayPassingClass', 'buildUpPlayPositioningClass',
       'chanceCreationPassing', 'chanceCreationPassingClass',
       'chanceCreationCrossing', 'chanceCreationCrossingClass',
       'chanceCreationShooting', 'chanceCreationShootingClass',
       'chanceCreationPositioningClass', 'defencePressure',
       'defencePressureClass', 'defenceAggression',
'defenceAggressionClass',
       'defenceTeamWidth', 'defenceTeamWidthClass',
       'defenceDefenderLineClass'
]

# Extracting the relevant data for clustering
team_performance_data =
Team_Attributes_with_Names[team_selected_columns]

# Getting numerical columns
numerical_columns =
team_performance_data.select_dtypes(include=['number']).columns.tolist
()

# Initializing figure
fig = go.Figure()

# Adding initial distribution trace for the first numerical column
initial_column = numerical_columns[0]
fig.add_trace(
    ff.create_distplot(
        [Team_Attributes_with_Names[initial_column].dropna()],
        [initial_column],
        bin_size=0.5,
        show_rug=False
    ).data[0]  # Get the first trace
)

# Creating the dropdown buttons for interactive plot
buttons = []
for column in numerical_columns:
```

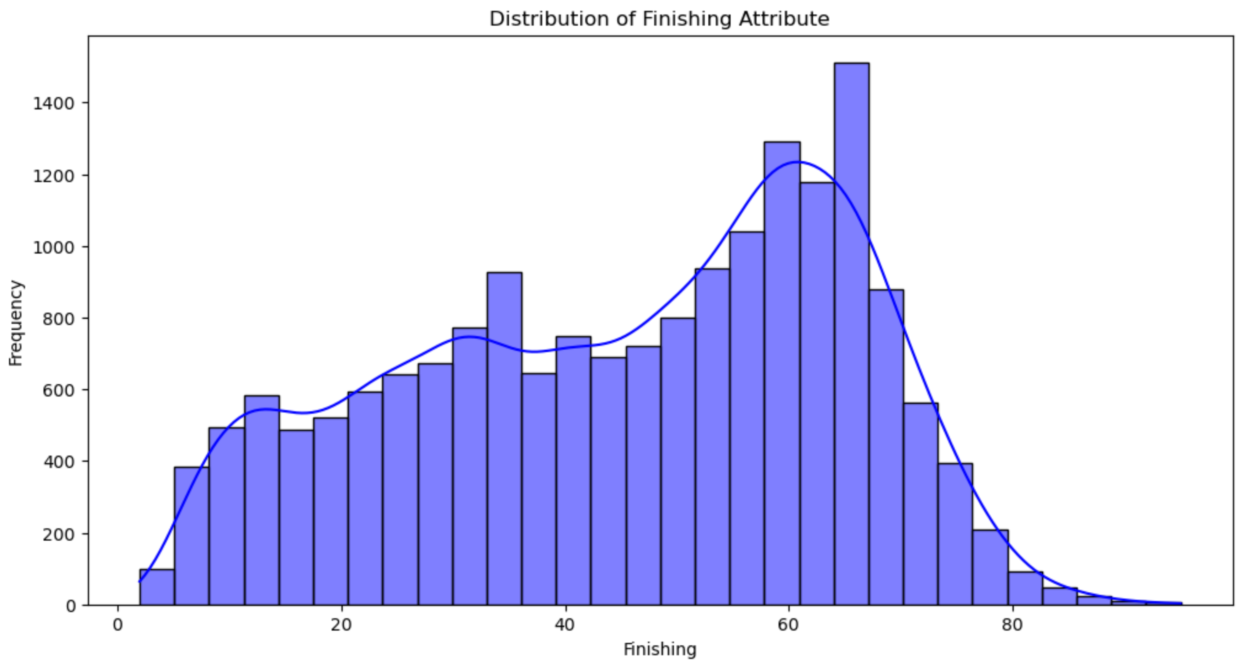
```
    buttons.append(
        dict(
            method='update',
            label=column,
            args=[{'x': [Team_Attributes_with_Names[column].dropna()],
                   'name': column},
                  {'title': f'Distribution of {column}'}]
        )
    )

# Updating the layout with dropdown
fig.update_layout(
    updatemenus=[dict(
        active=0,
        buttons=buttons,
        direction="down",
        pad={"r": 10, "t": 10},
        showactive=True,
        x=0.0,
        xanchor="left",
        y=1.15,
        yanchor="top"
    )],
    title_text=f'Distribution of {initial_column}',
    xaxis_title=initial_column,
    yaxis_title='Frequency',
    template='plotly_white'
)


fig.show()
```



What is the distribution of players' finishing skills, and how does it vary across different skill levels?

```
# Distribution plot for Finishing
plt.figure(figsize=(12, 6))
sns.histplot(Player_Attributes_with_Names['finishing'], bins=30,
kde=True, color='blue')
plt.title('Distribution of Finishing Attribute')
plt.xlabel('Finishing')
```

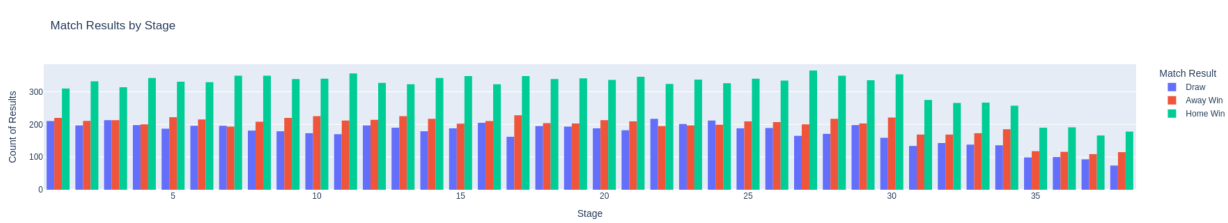
```
plt.ylabel('Frequency')
plt.show()
```



Bivariate Analysis

How are match results distributed across different stages of the season?

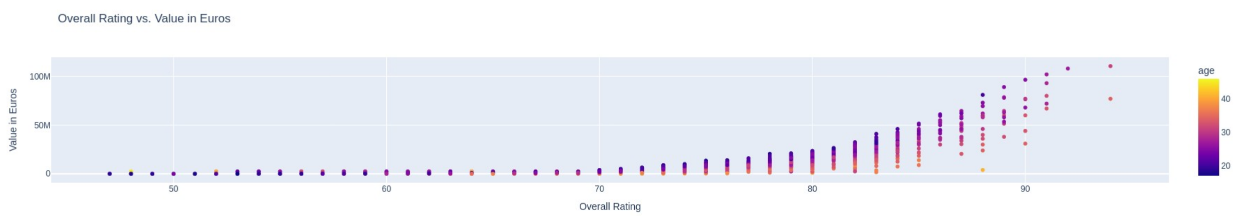
```
# Matching the results distribution by stage
fig = px.histogram(MCLT_Combined, x='stage', color='match_result',
barmode='group',
                   title="Match Results by Stage")
fig.update_layout(xaxis_title="Stage", yaxis_title="Count of Results",
legend_title="Match Result")
fig.show()
```



How does a player's overall rating correlate with their market value in euros?

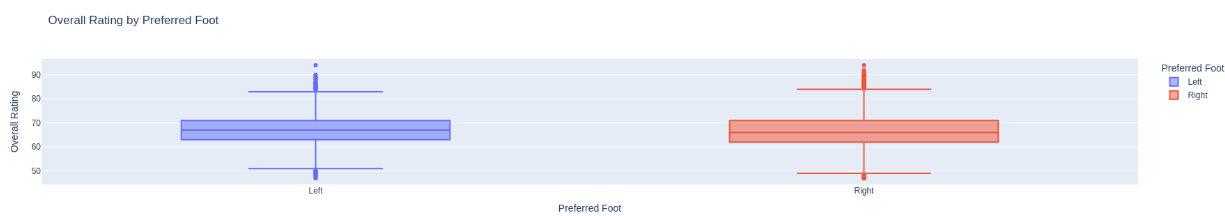
```
# Scatter plot of Overall Rating vs. Value in Euros
fig = px.scatter(Player_Attributes_with_Names,
                 x='overall_rating',
```

```
                  y='value_euro',
                  hover_name='name',
                  title='Overall Rating vs. Value in Euros',
                  labels={'overall_rating': 'Overall Rating',
'value_euro': 'Value in Euros'},
                  color='age',
                  template='plotly')
fig.show()
```



How does a player's overall rating vary depending on their preferred foot?

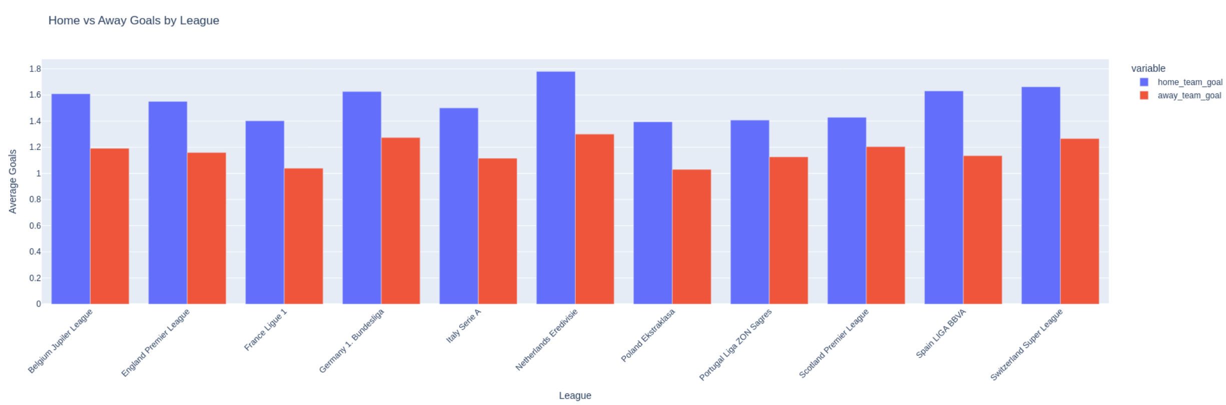
```
# Box plot of Overall Rating by Preferred Foot
fig = px.box(Player_Attributes_with_Names,
             x='preferred_foot',
             y='overall_rating',
             color='preferred_foot',
             title='Overall Rating by Preferred Foot',
             labels={'preferred_foot': 'Preferred Foot',
'overall_rating': 'Overall Rating'},
             template='plotly')
fig.show()
```



How do average home goals compare to average away goals across different leagues?

```
# Calculating the mean home and away goals by league
home_goals = MCLT_Combined.groupby('name_league')
['home_team_goal'].mean().reset_index()
away_goals = MCLT_Combined.groupby('name_league')
['away_team_goal'].mean().reset_index()

# Merging home and away goals into one DataFrame
merged_goals = home_goals.merge(away_goals, on='name_league',
suffixes=('_home', '_away'))
```

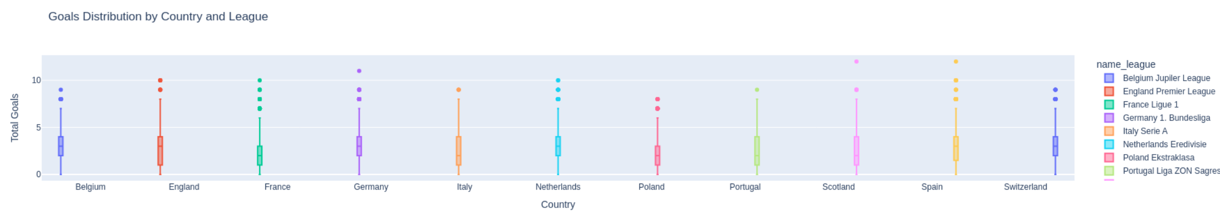
```python
# Creating the bar chart using Plotly
fig = px.bar(
    merged_goals,
    x='name_league',
    y=['home_team_goal', 'away_team_goal'],
    labels={'value': 'Average Goals', 'name_league': 'League'},  # Adding axis labels
    title='Home vs Away Goals by League'
)

fig.update_layout(
    barmode='group',
    xaxis_tickangle=-45,
    height=600,
    width=1000
)

fig.show()
```



Multivariate Analysis

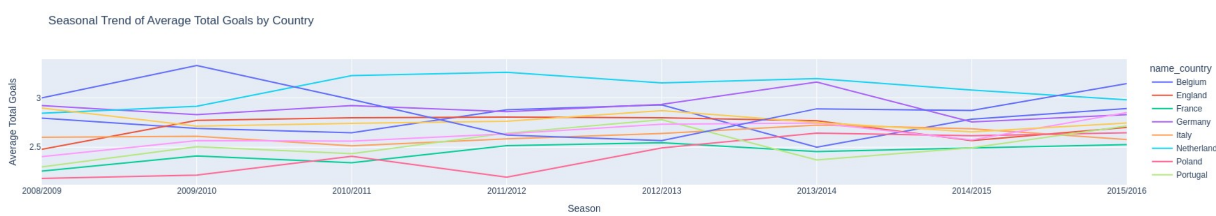
How do the total goals differ across countries and leagues?

```python
# Total goals by country and league
fig = px.box(MCLT_Combined, x="name_country", y="total_goals", color="name_league",
             title="Goals Distribution by Country and League")
fig.update_layout(xaxis_title="Country", yaxis_title="Total Goals")
fig.show()
```



How does the average total number of goals scored per season vary by country collectively ?

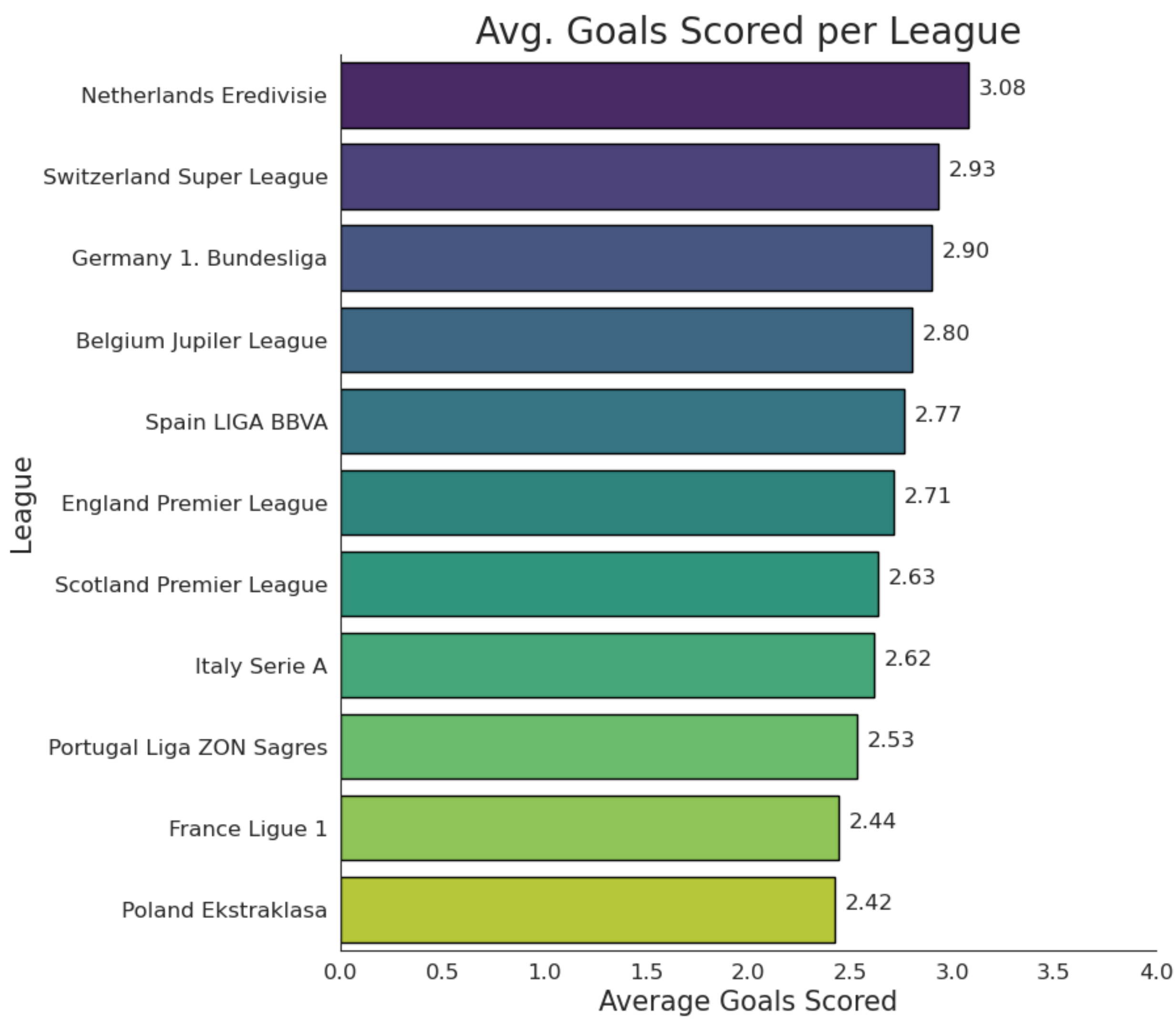
```
# Line plot of total goals per season by country
total_goals_trend = MCLT_Combined.groupby(['season', 'name_country'])
['total_goals'].mean().reset_index()

fig_goals_trend = px.line(total_goals_trend, x='season',
y='total_goals', color='name_country',
                          title='Seasonal Trend of Average Total Goals
by Country')
fig_goals_trend.update_layout(yaxis_title='Average Total Goals',
xaxis_title='Season')
fig_goals_trend.show()
```



What are the average goals scored per league

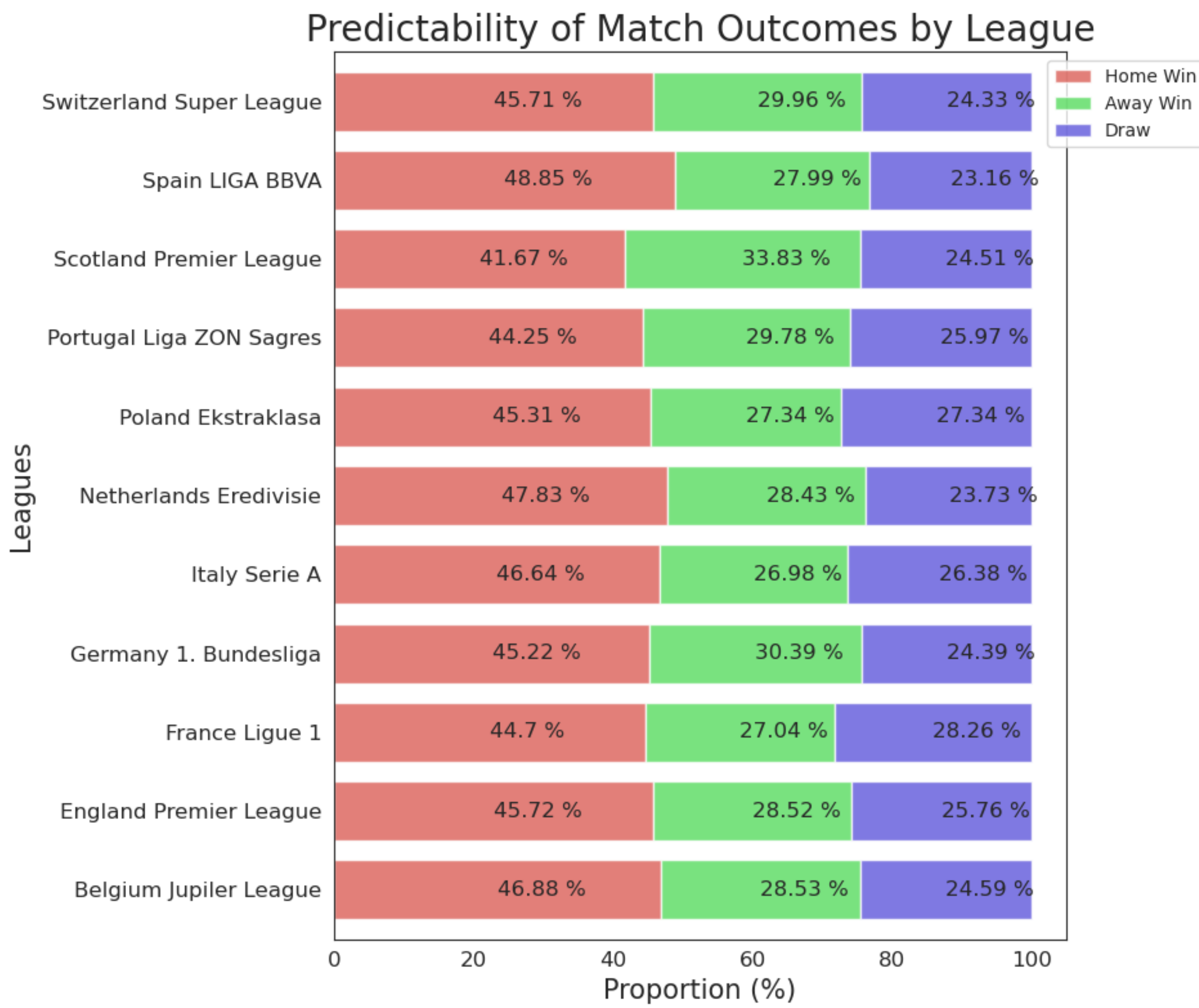
```
# Converting the 'total_goals' column to numeric
MCLT_Combined['total_goals'] =
pd.to_numeric(MCLT_Combined['total_goals'], errors='coerce')

# Calculating the Average Goals per League
LeagueAvgGoal_df = MCLT_Combined.groupby(['name_league'])
['total_goals'].mean().sort_values(ascending=False).reset_index()

# Plotting the average goals scored per league
sns.set_style('white')

# Defining a color palette
color_palette = sns.color_palette("viridis", len(LeagueAvgGoal_df))

# Creating the bar plot
a = sns.catplot(kind='bar', x='total_goals', y='name_league',
data=LeagueAvgGoal_df,
                ci='sd', edgecolor='k', palette=color_palette)
```

```python
# Customizing plot labels
a.fig.set_size_inches(12, 8)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Average Goals Scored', fontsize=15)
plt.ylabel('League', fontsize=15)
plt.xlim(0, 4)  # Adjust as necessary for your data
plt.title('Avg. Goals Scored per League', fontsize=20)

# Annotating bars with their values
for index, value in enumerate(LeagueAvgGoal_df['total_goals']):
    plt.text(value + 0.05, index, f"{value:.2f}", fontsize=12)  #
Adjust 0.05 for spacing

plt.show()
```



What are the proportions of match outcomes (home win, away win, draw) for each league, and how predictable are the match results across different leagues?

```
# Creating the pivot table
LeagueProp = MCLT_Combined.pivot_table(index='name_league',
columns='match_result',
                                           values='match_api_id',
aggfunc='count', fill_value=0)

# Summing up the total number of matches played in each league
LeagueProp['sum'] = LeagueProp.sum(axis=1)

result_columns = ['Home Win', 'Away Win', 'Draw']

# Calculating the proportions
LeagueProp = LeagueProp[result_columns].divide(LeagueProp['sum'],
axis=0).multiply(100)

# Plotting the proportions
ax = LeagueProp.plot.barh(stacked=True, figsize=(10, 8),
                           width=0.75, color=sns.color_palette("hls",
len(result_columns)),
                           edgecolor='w', alpha=0.8)

# Customizing the plot
ax.legend(result_columns, bbox_to_anchor=(1.2, 1), loc='upper right')
plt.title('Predictability of Match Outcomes by League', fontsize=20)
plt.xlabel('Proportion (%)', fontsize=15)
plt.ylabel('Leagues', fontsize=15)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Annotating each bar with the percentage values
for i, league in enumerate(LeagueProp.index):
    cumulative = 0
    for result in result_columns:
        cumulative += LeagueProp.loc[league, result]
        plt.text(cumulative - LeagueProp.loc[league, result] / 2, i,
                 f"{round(LeagueProp.loc[league, result], 2)} %",
fontsize=12, va='center')


plt.tight_layout()
plt.show()
```

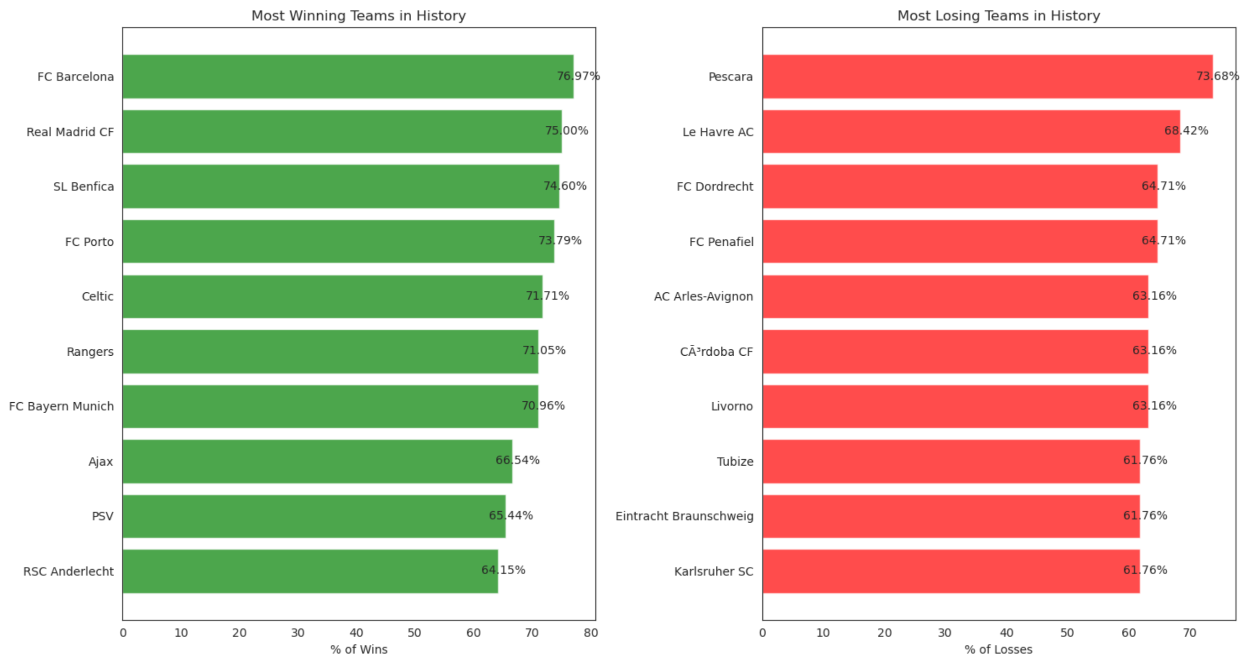


Which teams have the highest win and loss percentages in historical matches, and how do these metrics compare across the top teams?

```
MCLT_Combined.columns

Index(['id', 'country_id', 'league_id', 'season', 'stage', 'date',
       'match_api_id', 'home_team_api_id', 'away_team_api_id',
       'home_team_goal', 'away_team_goal', 'name_league',
'name_country',
       'home_team_long_name', 'away_team_long_name', 'match_result',
       'total_goals', 'goal_difference'],
      dtype='object')

df = MCLT_Combined.copy()

# Calculating the win percentage for each team
home_wins = df[df['match_result'] == 'Home Win']
['home_team_long_name'].value_counts()
away_wins = df[df['match_result'] == 'Away Win']
```

```
['away_team_long_name'].value_counts()
total_wins = home_wins.add(away_wins, fill_value=0)

home_losses = df[df['match_result'] == 'Away Win']
['home_team_long_name'].value_counts()
away_losses = df[df['match_result'] == 'Home Win']
['away_team_long_name'].value_counts()
total_losses = home_losses.add(away_losses, fill_value=0)

total_matches_home = df['home_team_long_name'].value_counts()
total_matches_away = df['away_team_long_name'].value_counts()
total_matches = total_matches_home.add(total_matches_away,
fill_value=0)

# Win and Loss percentages
win_percentage = (total_wins / total_matches) * 100
loss_percentage = (total_losses / total_matches) * 100

# Getting top 10 winning teams
top_winning_teams =
win_percentage.sort_values(ascending=False).head(10)

# Getting top 10 losing teams
top_losing_teams =
loss_percentage.sort_values(ascending=False).head(10)

# Plotting the results
fig, ax = plt.subplots(1, 2, figsize=(15, 8))

# Most winning teams
ax[0].barh(top_winning_teams.index, top_winning_teams.values,
color='green', alpha=0.7)
ax[0].set_title('Most Winning Teams in History')
ax[0].set_xlabel('% of Wins')
ax[0].invert_yaxis()

# Most losing teams
ax[1].barh(top_losing_teams.index, top_losing_teams.values,
color='red', alpha=0.7)
ax[1].set_title('Most Losing Teams in History')
ax[1].set_xlabel('% of Losses')
ax[1].invert_yaxis()

# Adding % to bars
for i in ax:
    for bar in i.patches:
        i.annotate(f"{bar.get_width():.2f}%",
                   (bar.get_width(), bar.get_y() +
bar.get_height()/2),
                   ha='center', va='center', xytext=(5, 0),
```

```
                    textcoords='offset points')

# Displaying the plots
plt.tight_layout()
plt.show()
```



Dynamic Dropdown

How do the average home and away goals scored by teams vary across different seasons and leagues?

```
# Creating a pivot table to show average home and away goals by season
and league
performance_pivot =
MCLT_Combined.pivot_table(values=['home_team_goal', 'away_team_goal'],
                                                 index=['season',
'name_league'], aggfunc='mean').reset_index()

# Initializing the figure with subplots
fig_home_away_performance = make_subplots(rows=1, cols=1)

# Listing of unique leagues for the dropdown
leagues = performance_pivot['name_league'].unique()

# Adding traces for each league
for league in leagues:
    # Filter data for each league
    league_data = performance_pivot[performance_pivot['name_league']
== league]
```

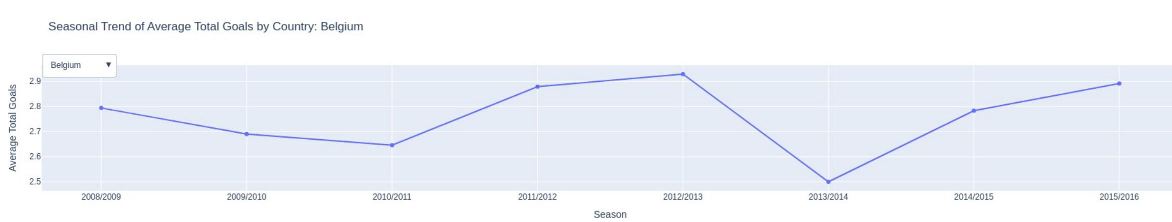
```
    # Home goals trace
    fig_home_away_performance.add_trace(
        go.Scatter(
            x=league_data['season'],
            y=league_data['home_team_goal'],
            name=f'Home Goals - {league}',
            visible=(league == leagues[0])  # Only the first trace is
visible by default
        ),
        row=1, col=1
    )

    # Away goals trace
    fig_home_away_performance.add_trace(
        go.Scatter(
            x=league_data['season'],
            y=league_data['away_team_goal'],
            name=f'Away Goals - {league}',
            visible=(league == leagues[0])  # Only the first trace is
visible by default
        ),
        row=1, col=1
    )

# Creating dropdown buttons for interactive plot
buttons = []
for league in leagues:
    buttons.append(
        dict(
            method='update',
            label=league,
            args=[{'visible': [(league == l) or (league ==
l.replace('Home', 'Away')) for l in leagues]},
                  {'title': f'Home vs Away Team Performance:
{league}'}]
        )
    )

# Updating layout with dropdown
fig_home_away_performance.update_layout(
    updatemenus=[
        dict(
            active=0,
            buttons=buttons,
            direction="down",
            pad={"r": 10, "t": 10},
            showactive=True,
            x=0.0,
            xanchor="left",
```

```
            y=1.15,
            yanchor="top"
        ),
    ],
    title_text='Home vs Away Team Performance by Season and League',
    yaxis_title='Average Goals',
    xaxis_title='Season'
)

# Setting y-axis range for uniformity
fig_home_away_performance.update_yaxes(range=[0,
max(performance_pivot['home_team_goal'].max(),
performance_pivot['away_team_goal'].max())])

fig_home_away_performance.show()
```

How does the average total number of goals scored per season vary by country?

```
# Aggregating the data
total_goals_trend = MCLT_Combined.groupby(['season', 'name_country'])
['total_goals'].mean().reset_index()

# Creating a unique line plot for each country
fig_goals_trend = go.Figure()

# Listing of all unique countries
countries = total_goals_trend['name_country'].unique()

# Adding traces for each country, but only one will be visible at a
time based on the dropdown selection
for country in countries:
    visible = True if country == countries[0] else False
    filtered_data =
total_goals_trend[total_goals_trend['name_country'] == country]
    fig_goals_trend.add_trace(
        go.Scatter(
            x=filtered_data['season'],
            y=filtered_data['total_goals'],
            name=country,
            visible=visible
        )
```

```
    )

# Creating dropdown buttons that allow choosing which country's data
to display
buttons = []
for country in countries:
    buttons.append(
        dict(
            method='update',
            label=country,
            args=[{'visible': [country == k for k in countries]},
                  {'title': f'Seasonal Trend of Average Total Goals by
Country: {country}'}]
        )
    )

# Including dropdown
fig_goals_trend.update_layout(
    updatemenus=[
        dict(
            active=0,
            buttons=buttons,
            direction="down",
            pad={"r": 10, "t": 10},
            showactive=True,
            x=0.0,
            xanchor="left",
            y=1.15,
            yanchor="top"
        ),
    ],
    yaxis_title='Average Total Goals',
    xaxis_title='Season',
    title=f'Seasonal Trend of Average Total Goals by Country:
{countries[0]}'
)

fig_goals_trend.show()
```



Insights

Following are the key findings that were derived from EDA

Visualization	Key Finding
Mean Overall Rating by Age Group	Players aged **25-30** have the highest mean overall rating, indicating peak performance in this age range.
Total Value in Euros by Age Group	Players in the **20-25** and **25-30** age groups have the highest total market value, highlighting prime investment years.
Distribution of BMI Categories	The majority of players fall into the **normal weight** category, with fewer classified as overweight, underweight, or obese.
Count of Players by Offensive Score Category	Most players have **medium or high offensive scores**, indicating a well-distributed offensive capability across players.
Count of Players by Defensive Score Category	The majority of players have **medium defensive scores**, with very few achieving high defensive scores.
Match Count by League	The **England Premier League**, **France Ligue 1**, and **Italy Serie A** have the most recorded matches in the dataset.
Match Result Distribution	**Home wins** are significantly more common than away wins or draws, indicating a strong home-field advantage in football.
Distribution of Goal Difference (Home vs Away)	Home wins have larger positive goal differences, while away wins show smaller negative goal differences.
Distribution of Home Team Goals	Most home teams score between **0-2 goals**, with very few teams scoring more than 4 goals.
Distribution of Away Team Goals	Most away teams score **0-1 goals**, with higher scores being much rarer than home team scores.
Distribution of Finishing Attribute	The finishing attribute is most frequently clustered between **40 and 60**, with a gradual drop-off beyond 60.

Visualization	Key Finding
Avg. Goals Scored per League	The **Netherlands Eredivisie** has the highest average goals per match (3.08), suggesting a high-scoring league.
Predictability of Match Outcomes by League	**Spain LIGA BBVA** has the highest proportion of home wins (48.85%), showing a strong home-field advantage.
Most Winning Teams in History	**FC Barcelona** has the highest win percentage (76.97%) among football teams, followed by **Real Madrid**.
Most Losing Teams in History	**Pescara** has the highest percentage of losses (73.68%) in football history, followed by **Le Havre AC**.
Match Results by Stage	Home wins are the most frequent match outcome across all stages, indicating a consistent home advantage.
Overall Rating by Preferred Foot	Players who are right-footed have a slightly higher overall rating compared to left-footed players.
Home vs Away Goals by League	Across leagues, home teams consistently score more goals on average compared to away teams, reinforcing home advantage.
Goals Distribution by Country and League	Leagues in England and Spain have a higher spread in total goals scored compared to other countries.
Seasonal Trend of Average Total Goals by Country	The Netherlands consistently shows a high average of goals per season, while other countries exhibit fluctuating trends.

Hypothesis Testing

Hypothesis Testing is a statistical method used to determine whether there is enough evidence in a sample of data to support a specific claim or hypothesis about a population parameter. It involves formulating a null hypothesis (H_0) and an alternative hypothesis (H_1), and then using statistical tests to evaluate the likelihood of the observed data under the null hypothesis.

Skewness

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable. Positive skewness indicates a longer right tail, while negative skewness indicates a longer left tail. A skewness of zero suggests a symmetrical distribution.

Interpretation of Skewness Values:

- Approximately Symmetric (T-test-friendly): Skewness between -0.5 and 0.5
- Moderately Skewed (Consider transformations): Skewness between -1 and -0.5 or 0.5 and 1
- Highly Skewed (Use non-parametric tests or transformations): Skewness below -1 or above 1

```
# Automatically select integer columns
int_columns =
Player_Attributes_with_Names.select_dtypes(include=['int64']).columns
int_columns = int_columns.drop('age')

# Calculating the skewness
skewness = Player_Attributes_with_Names[int_columns].apply(lambda x:
skew(x.dropna()))

# Displaying the skewness
print("Skewness of Integer Variables:")
print(skewness)
```

```
Skewness of Integer Variables:
overall_rating                 0.046194
potential                      0.262984
international_reputation(1-5)    4.105136
weak_foot(1-5)                 0.149423
skill_moves(1-5)               0.147877
crossing                      -0.597419
finishing                     -0.293019
heading_accuracy              -0.883726
short_passing                 -1.095371
volleys                       -0.148474
dribbling                     -1.079623
curve                         -0.240741
freekick_accuracy              0.109290
long_passing                  -0.592150
ball_control                  -1.253168
```

```
acceleration                    -0.848305
sprint_speed                    -0.882557
agility                         -0.598371
reactions                       -0.123551
balance                         -0.579763
shot_power                      -0.667196
jumping                         -0.442933
stamina                         -0.940101
strength                        -0.461787
long_shots                      -0.416374
aggression                      -0.432910
interceptions                   -0.273768
positioning                     -0.722056
vision                          -0.350344
penalties                       -0.345322
composure                       -0.417117
marking                         -0.368186
standing_tackle                 -0.344697
sliding_tackle                  -0.274019
offensive_score                 -0.796612
defensive_score                 -0.422504
dtype: float64
```

Nearly Symmetric:

Variables such as overall_rating (0.046), potential (0.26), weak_foot (0.15), and skill_moves (0.15) have skewness values close to 0, suggesting a near-symmetric distribution, meaning the majority of values are centered around the mean.

Highly Positively Skewed:

- international_reputation (4.1) shows a significant positive skew, indicating that most players have a lower international reputation, while a few have very high values.

  This suggests that elite players with high reputations are outliers compared to the general player population.

Moderately Negative Skew:

- Attributes like short_passing (-1.10), dribbling (-1.08), ball_control (-1.25), and stamina (-0.94) show moderate negative skewness, suggesting a concentration of players with higher values in these skills and a longer tail on the lower end of the distribution.

  This could indicate that a significant proportion of players are relatively strong in these attributes, with fewer players exhibiting poor performance in these skills.

T-Test

A T-Test is a statistical test used to compare the means of two groups to determine if they are significantly different from each other. It assumes that the data follows a normal distribution and is often used when the sample size is small.

For T-tests (Normally Distributed Variables):

- H0: The means of the two groups are equal.
- H1: The means of the two groups are different.

Q. Is there a significant difference in the average overall rating between left-footed and right-footed players?

> Null Hypothesis (H0): The average overall rating is the same for left-footed and right-footed players.
>
> Alternative Hypothesis (H1): The average overall rating is different between left-footed and right-footed players.

If the p-value is less than the significance level (alpha = 0.05), we reject the null hypothesis and conclude that there is a significant difference in the average overall rating between the two groups. Otherwise, we fail to reject the null hypothesis.

```
Player_Attributes_with_Names.preferred_foot.unique()
```

```
array(['Left', 'Right'], dtype=object)
```

```
# Grouping the data
left_footed = 
Player_Attributes_with_Names[Player_Attributes_with_Names['preferred_f
oot'] == 'Left']['overall_rating'].dropna()
right_footed = 
Player_Attributes_with_Names[Player_Attributes_with_Names['preferred_f
oot'] == 'Right']['overall_rating'].dropna()

# Performing T-test (independent samples T-test)
t_stat, p_value = stats.ttest_ind(left_footed, right_footed)

# Printing the results
print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Setting significance level
alpha = 0.05

# Conclusion based on p-value
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant 
difference in the average overall rating between left-footed and 
right-footed players.")
```

```
else:
    print("Fail to reject the null hypothesis: There is no significant difference in the average overall rating between left-footed and right-footed players.")
```

```
T-statistic: 6.4057
P-value: 0.0000
Reject the null hypothesis: There is a significant difference in the average overall rating between left-footed and right-footed players.
```

Mann-Whitney U Test

The Mann-Whitney U Test is a non-parametric test that compares the distributions of two independent samples. It assesses whether one sample tends to have larger values than the other, making it useful when the assumption of normality is not met.

For Mann-Whitney U Tests (Skewed Variables):

- H0: The distributions of the two groups are the same.
- H1: The distributions of the two groups are different.

Q. Is there a significant difference in acceleration between younger players (e.g., <25 years old) and older players (e.g., >=25 years old)?

> Null Hypothesis (H0): The distribution of acceleration is the same for younger and older players.
>
> Alternative Hypothesis (H1): The distribution of acceleration is different between younger and older players.

If the p-value is less than the significance level (alpha = 0.05), we reject the null hypothesis and conclude that there is a significant difference in acceleration between younger and older players. Otherwise, we fail to reject the null hypothesis.

```
Player_Attributes_with_Names.age_category.unique()
```

```
['30+', '25-30', '20-25', 'Under 20']
Categories (4, object): ['Under 20' < '20-25' < '25-30' < '30+']
```

```
younger_players = Player_Attributes_with_Names[Player_Attributes_with_Names['age_category'].isin(['Under 20', '20-25'])]['acceleration'].dropna()
older_players = Player_Attributes_with_Names[Player_Attributes_with_Names['age_category'].isin(['25-30', '30+'])]['acceleration'].dropna()

# Performing Mann-Whitney U test
u_stat, p_value = mannwhitneyu(younger_players, older_players)

# Printing the results
print(f"U-statistic: {u_stat:.4f}")
```

```python
print(f"P-value: {p_value:.4f}")

# Setting significance level
alpha = 0.05

# Conclusion based on p-value
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in acceleration between younger and older players.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in acceleration between younger and older players.")
```

```
U-statistic: 44764634.0000
P-value: 0.0000
Reject the null hypothesis: There is a significant difference in
acceleration between younger and older players.
```

Chi-Square Test

A Chi-Square Test is a statistical test used to determine if there is a significant association between categorical variables. It compares the observed frequencies in each category to the frequencies expected under the null hypothesis, helping to identify relationships or differences among groups.

> The Chi-Square test of independence is used to determine whether there is a significant association between two categorical variables.

Q1. Is there an association between a team's build-up play strategy (i.e., buildUpPlaySpeedClass) and their match result (i.e, win, draw, or loss)?

> Null Hypothesis (H0): There is no association between a team's defensive pressure class and the match result.
>
> Alternative Hypothesis (H1): There is an association between a team's defensive pressure class and the match result.

```python
# Joining Team_Attributes_with_Names and MCLT_Combined on the  team ID
team_match_data = pd.merge(Team_Attributes_with_Names, MCLT_Combined,
left_on='team_api_id', right_on='home_team_api_id')

# Creating a contingency table for buildUpPlaySpeedClass vs
match_result
contingency_table =
pd.crosstab(team_match_data['buildUpPlaySpeedClass'],
team_match_data['match_result'])

# Performing Chi-Square test
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Printing the results
```

```python
print("Chi-Square Statistic:", chi2)
print("P-value:", p_value)

# Setting the significance level
alpha = 0.05

# Conclusion based on p-value
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant 
association between build-up play speed class and match result.")
else:
    print("Fail to reject the null hypothesis: There is no significant 
association between build-up play speed class and match result.")
```

```
Chi-Square Statistic: 116.84888445707327
P-value: 2.5150825608310044e-24
Reject the null hypothesis: There is a significant association between 
build-up play speed class and match result.
```

Q2. Is there an association between a team's chance creation strategy (i.e., chanceCreationShootingClass) and the total number of goals scored in the match?

> Null Hypothesis (H0): There is no association between a team's chance creation shooting class and the total number of goals scored in a match. Alternative Hypothesis (H1): There is an association between a team's chance creation shooting class and the total number of goals scored.

```python
# Creating a contingency table for chanceCreationShootingClass vs 
total_goals
contingency_table_shooting = 
pd.crosstab(team_match_data['chanceCreationShootingClass'], 
team_match_data['total_goals'])

# Performing Chi-Square test
chi2_shooting, p_value_shooting, dof_shooting, expected_shooting = 
chi2_contingency(contingency_table_shooting)

# Print results
print("Chi-Square Statistic:", chi2_shooting)
print("P-value:", p_value_shooting)

# Conclusion based on p-value
if p_value_shooting < alpha:
    print("Reject the null hypothesis: There is a significant 
association between chance creation shooting class and total goals 
scored.")
else:
    print("Fail to reject the null hypothesis: There is no significant 
association between chance creation shooting class and total goals 
scored.")
```

```
Chi-Square Statistic: 171.1277122503828
P-value: 3.57319154143563e-24
Reject the null hypothesis: There is a significant association between
chance creation shooting class and total goals scored.
```

Insights

At th end of Hypothesis Testing section we derived the following insights,

From Chi-Square:

- There is a significant association between chance creation shooting class and total goals scored.
- There is a significant association between build-up play speed class and match result.

From T-Test:

- There is a significant difference in the average overall rating between left-footed and right-footed players.

From Mann-Whitney U Test:

- There is a significant difference in acceleration between younger and older players.

Cluster Analysis

Cluster analysis is an unsupervised machine learning technique used to group similar data points based on their features. In this project, we applied various clustering methods, such as K-Means, Gaussian Mixture Models (GMM), and Fuzzy C-Means, to identify player archetypes and team patterns. By clustering players based on attributes like dribbling, finishing, and ball control, we aimed to uncover hidden groupings that reveal distinct player roles and skill sets. Additionally, we used PCA (Principal Component Analysis) for dimensionality reduction, allowing us to visualize these clusters in lower-dimensional spaces for better interpretation and insight.

Optimal n

"optimal n" typically refers to determining the best number of clusters (denoted as n) for clustering algorithms.

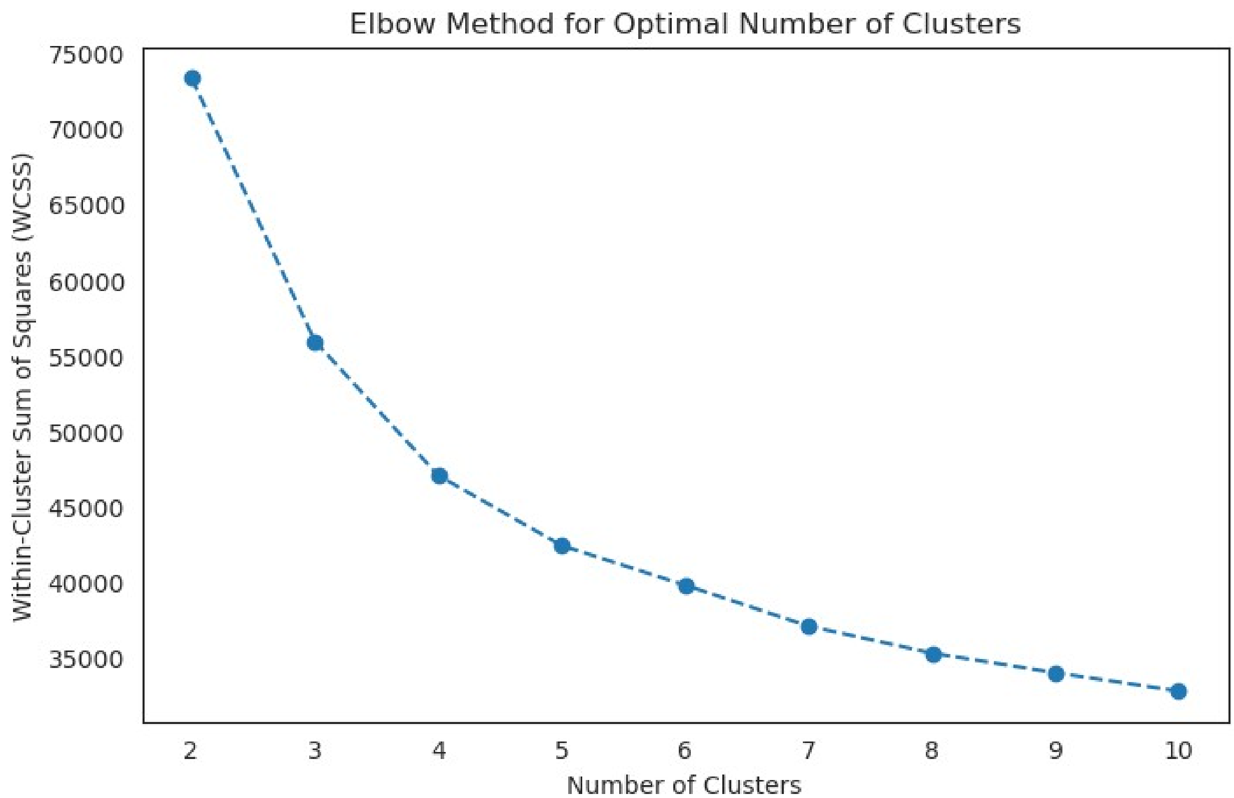
Methods to Identify the Optimal Number of Clusters:

1. Elbow Method: Concept: The Elbow method helps to identify the number of clusters by computing the Within-Cluster Sum of Squares (WCSS) for different values of n (number of clusters). WCSS measures the variance within each cluster, and the goal is to minimize it. The optimal number of clusters is chosen where the WCSS starts to flatten out (i.e., where adding more clusters doesn’t significantly reduce the WCSS). Interpretation: The "elbow" point on the plot indicates the optimal number of clusters.
2. Silhouette Score: Concept: The Silhouette score measures how similar an object is to its own cluster compared to other clusters. The score ranges from -1 to 1, where a higher

score indicates that clusters are well separated and dense. Interpretation: The optimal number of clusters is usually where the Silhouette score is maximized. Approach: Both the Elbow method and Silhouette score can be calculated before running any specific clustering technique like K-Means, GMM, etc., and they help guide you toward the best value for n.

Elbow Method

```
# Selecting the relevant numerical features for analysis
features = ['dribbling', 'finishing', 'crossing', 'ball_control',
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values

# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Defining the range of clusters to evaluate
range_n_clusters = list(range(2, 11))

# Elbow Method - Calculating WCSS (Within-Cluster Sum of Squares)
wcss = []
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow method results
plt.figure(figsize=(8, 5))
plt.plot(range_n_clusters, wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.show()
```



In the Elbow plot, we see a distinct bend around n=4. This suggests that adding more clusters beyond 4 results in diminishing returns in terms of reducing the Within-Cluster Sum of Squares (WCSS). The curve flattens after 4, indicating that while adding more clusters might slightly reduce the WCSS, the improvement is marginal compared to the jump between 3 and 4 clusters. The optimal number of clusters could be n=4, as the "elbow" is clearly visible at this point.

Silhouette Method

```
# Silhouette Method - Calculating Silhouette Scores
silhouette_avg = []
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X_scaled)
    silhouette_avg.append(silhouette_score(X_scaled, cluster_labels))

# Plotting the Silhouette score results
plt.figure(figsize=(8, 5))
plt.plot(range_n_clusters, silhouette_avg, marker='o', linestyle='--')
plt.title('Silhouette Score for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```

The Silhouette score plot shows the highest score for n=2, but the score drops significantly after that. The Silhouette score at n=3 and n=4 is lower but still within a reasonable range. After that, the score continues to decline slightly.

Given that:

The Elbow method strongly suggests n=4 as the point where adding more clusters yields diminishing returns. The Silhouette score shows a clear drop after n=2, but n=3 and n=4 still provide relatively decent scores (though not as high as n=2).

K means

K-Means is a popular unsupervised machine learning algorithm used for clustering data into distinct groups based on feature similarities. The algorithm partitions the dataset into k clusters, where each data point belongs to the cluster with the nearest mean (centroid). K-Means aims to minimize the within-cluster variance, making clusters as compact as possible.

To have Clustering Method Visualization Flexibility we will explore performing K means with PCA in the following combinations, the benefits of using them is also stated below,

- K-Means without PCA - Limited to 2D/3D plots of selected features like dribbling vs finishing, but fully interpretable.
- K-Means with PCA for visualization - Excellent for visualizing clusters in 2D or 3D space, but loses feature-specific interpretability.
- PCA first, then K-Means on PCA-reduced data - Best for handling large/high-dimensional datasets, but visualizing clusters is less intuitive.

What are the distinct groups or clusters of players based on their attributes ?

Without PCA

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing', 'crossing', 'ball_control',
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values

# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying K-Means Clustering
kmeans = KMeans(n_clusters=4, random_state=42)  # 4 clusters
Player_Attributes_with_Names['cluster_kmeans'] =
kmeans.fit_predict(X_scaled)

#  Visualizing the clusters using two original features
fig = px.scatter(
    Player_Attributes_with_Names,
    x='dribbling', y='finishing',
```

```
    color='cluster_kmeans',
    hover_data=['full_name'],  # Show player names on hover
    title='K-Means Clustering: Dribbling vs Finishing',
    color_continuous_scale=px.colors.qualitative.Bold
)

fig.show()
```

PCA for Cluster visualization

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing', 'crossing', 'ball_control',
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values

# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying K-Means Clustering
kmeans = KMeans(n_clusters=4, random_state=42)  # 4 clusters
Player_Attributes_with_Names['cluster_kmeans'] =
kmeans.fit_predict(X_scaled)

# Applying PCA to reduce to 2 components for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualizing the clusters
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = Player_Attributes_with_Names['cluster_kmeans']
pca_df['full_name'] = Player_Attributes_with_Names['full_name']

# Ploting using Plotly
fig = px.scatter(
    pca_df,
    x='PC1', y='PC2',
    color='cluster',
    hover_data=['full_name'],
    title='K-Means Clustering of Players (2D PCA Visualization)',
    color_continuous_scale=px.colors.qualitative.Bold
)
fig.show()
```

Clustering on PCA-transformed data

```
# Standardizing the data (as done earlier)
X_scaled = scaler.fit_transform(X)

# Applying the PCA to reduce the dimensions to 2 components
```

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Applying K-Means Clustering on PCA-transformed data
kmeans_pca = KMeans(n_clusters=4, random_state=42)
Player_Attributes_with_Names['cluster_kmeans_pca'] =
kmeans_pca.fit_predict(X_pca)

# Creating a DataFrame for PCA components and clusters
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = Player_Attributes_with_Names['cluster_kmeans_pca']
pca_df['full_name'] = Player_Attributes_with_Names['full_name']

#  Visualizing the K-Means Clusters with PCA using Plotly
fig = px.scatter(
    pca_df,
    x='PC1', y='PC2',
    color='cluster',
    hover_data=['full_name'],
    title='K-Means Clustering with PCA (2D Visualization)',
    color_continuous_scale=px.colors.qualitative.Bold
)
fig.show()
```

Gaussian Mixture Models (GMM)

Gaussian Mixture Models (GMM) is a probabilistic clustering method that allows for soft clustering, meaning players can belong to multiple clusters with varying probabilities. This method is useful for identifying overlapping clusters and versatile players who may fit into multiple roles.

Steps for GMM Clustering:

- Standardize the data.
- Fit GMM to the data.
- Analyze the results (clusters, probabilities, etc.).
- Visualize the clusters.

How are soccer players grouped based on their dribbling and finishing skills when applying Gaussian Mixture Model clustering?

Without PCA

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing', 'crossing', 'ball_control',
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values

# Standardizing the data
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Applying Gaussian Mixture Model (GMM) for clustering
gmm = GaussianMixture(n_components=4, random_state=42)  # Adjust
n_components based on your data
Player_Attributes_with_Names['cluster_gmm'] =
gmm.fit_predict(X_scaled)

# Getting probabilities for each cluster (soft clustering)
Player_Attributes_with_Names['probabilities'] =
gmm.predict_proba(X_scaled).max(axis=1)

# Visualizing the clusters
fig = px.scatter(
    Player_Attributes_with_Names,
    x='dribbling', y='finishing',
    color='cluster_gmm',
    hover_data=['full_name', 'probabilities'],  # Show player names
and cluster probabilities on hover
    title='GMM Clustering of Players (Dribbling vs Finishing)',
    color_continuous_scale=px.colors.qualitative.Bold
)

# Displaying the plot
fig.show()
```

PCA for Cluster visualization

```
# Applying PCA to reduce the data to 2 dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Creating a DataFrame with PCA components and cluster assignments
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = Player_Attributes_with_Names['cluster_gmm']
pca_df['full_name'] = Player_Attributes_with_Names['full_name']
pca_df['probabilities'] =
Player_Attributes_with_Names['probabilities']

# Plotting the PCA-transformed data with Plotly
fig_pca = px.scatter(
    pca_df,
    x='PC1', y='PC2',
    color='cluster',
    hover_data=['full_name', 'probabilities'],
    title='GMM Clustering of Players (PCA Visualization)',
    color_continuous_scale=px.colors.qualitative.Bold
)

fig_pca.show()
```

Fuzzy C-Means

Fuzzy C-Means (FCM) clustering. FCM allows for soft clustering, meaning that each player can belong to multiple clusters with varying degrees of membership, much like GMM, but instead of using probabilistic membership, FCM uses fuzzy membership values. This method is useful when players can be considered part of multiple skill groups, much like versatile football players who may play in multiple positions.

How can we group football players into fuzzy skill-based clusters, and how strongly do players belong to each cluster, indicating potential versatility?

Without PCA

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing']
X = Player_Attributes_with_Names[features].values

# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying Fuzzy C-Means (FCM) clustering
n_clusters = 4  # Based on previous elbow and silhouette methods
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T, 
c=n_clusters, m=2, error=0.005, maxiter=1000)

# Getting cluster membership for each player (max membership)
cluster_membership = np.argmax(u, axis=0)
Player_Attributes_with_Names['fcm_cluster'] = cluster_membership

# Visualizing the clusters
fig = px.scatter(
    Player_Attributes_with_Names,
    x='dribbling', y='finishing',
    color='fcm_cluster',
    hover_data=['full_name'],  # Show player names on hover
    title='Fuzzy C-Means Clustering of Players (Dribbling vs 
Finishing)',
    color_continuous_scale=px.colors.qualitative.Bold
)

fig.show()
```

PCA for Cluster visualization

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing', 'crossing', 'ball_control', 
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values
```

```python
# Standardizing the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying Fuzzy C-Means (FCM) clustering
n_clusters = 4
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_scaled.T,
c=n_clusters, m=2, error=0.005, maxiter=1000)

#  Getting the cluster membership for each player (max membership)
cluster_membership = np.argmax(u, axis=0)
Player_Attributes_with_Names['fcm_cluster'] = cluster_membership

# Applying the PCA to reduce dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Creating a DataFrame with PCA components and cluster assignments
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = Player_Attributes_with_Names['fcm_cluster']
pca_df['full_name'] = Player_Attributes_with_Names['full_name']

# Visualizing the FCM clusters with PCA
fig_pca = px.scatter(
    pca_df,
    x='PC1', y='PC2',
    color='cluster',
    hover_data=['full_name'],
    title='Fuzzy C-Means Clustering of Players (PCA Visualization)',
    color_continuous_scale=px.colors.qualitative.Bold
)

fig_pca.show()
```

Self-Organizing Maps (SOM)

SOM is a type of neural network used for dimensionality reduction and clustering. It maps high-dimensional data into a lower-dimensional (usually 2D) grid while preserving the topology of the data.

SOM is great for visualizing high-dimensional player attributes in a 2D grid, making it easier to identify clusters of players.

Steps for SOM Visualization:

- Standardize the data: SOMs work better when the data is scaled, so we'll standardize the features before applying SOM.
- Train the SOM: We will train the SOM using MiniSom and assign each player (or team) to a specific grid location (called the "winning node").

- Visualize the SOM: We'll use color-coding to visualize clusters and distances between nodes.

How do soccer players cluster in the feature space defined by their attributes, as visualized by the Self-Organizing Map (SOM)?

```
# Selecting the relevant numerical features for clustering
features = ['dribbling', 'finishing', 'crossing', 'ball_control',
'short_passing', 'sprint_speed', 'strength', 'stamina']
X = Player_Attributes_with_Names[features].values

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initializing and training the SOM
som_grid_x, som_grid_y = 10, 10
som = MiniSom(x=som_grid_x, y=som_grid_y, input_len=X_scaled.shape[1],
sigma=1.0, learning_rate=0.5)

# Initializing random weights and train the SOM
som.random_weights_init(X_scaled)
som.train_random(data=X_scaled, num_iteration=100)

# Gathering the SOM output data for visualization
nodes_x = []
nodes_y = []
node_colors = []
hover_texts = []

# Assigning the colors to clusters
colors = px.colors.qualitative.Plotly
for i, x in enumerate(X_scaled):
    winning_node = som.winner(x)
    nodes_x.append(winning_node[0])
    nodes_y.append(winning_node[1])
    node_colors.append(colors[i % len(colors)])

hover_texts.append(Player_Attributes_with_Names['full_name'].iloc[i])

# Creating the U-Matrix (distance between nodes for the background
heatmap)
u_matrix = som.distance_map().T

# heatmap for U-Matrix
heatmap = go.Heatmap(
    z=u_matrix,
    x=np.arange(som_grid_x),
    y=np.arange(som_grid_y),
    colorscale='Greys',
```

```
    showscale=True,
    hoverinfo='none',  # Disable hover for the heatmap itself
)

# scatter plot for SOM nodes
scatter = go.Scatter(
    x=nodes_x,
    y=nodes_y,
    mode='markers',
    marker=dict(size=14, color=node_colors, line=dict(width=1,
color='black')),
    text=hover_texts,
    hoverinfo='text',
)

# Combining the heatmap and scatter plot into a single figure
layout = go.Layout(
    title="Self-Organizing Map (SOM) - Interactive Player Clustering",
    xaxis=dict(title='SOM X-Axis', range=[-0.5, som_grid_x - 0.5],
showgrid=False),
    yaxis=dict(title='SOM Y-Axis', range=[-0.5, som_grid_y - 0.5],
showgrid=False, scaleanchor='x', scaleratio=1),
    width=800,
    height=800,
)

fig = go.Figure(data=[heatmap, scatter], layout=layout)


fig.show()

# Number of players in the original dataset
total_players = len(Player_Attributes_with_Names)

# Number of players mapped to the SOM
mapped_players = len(nodes_x)

print(f"Total players in dataset: {total_players}")
print(f"Total players mapped to SOM: {mapped_players}")

node_assignments = {}

# Assigning the players to their respective nodes
for i, x in enumerate(X_scaled):
    winning_node = som.winner(x)
    if winning_node not in node_assignments:
        node_assignments[winning_node] = []

node_assignments[winning_node].append(Player_Attributes_with_Names['fu
ll_name'].iloc[i])
```

```python
# Checking the nodes and the players assigned to each
for node, players in node_assignments.items():
    print(f"Node {node}: {players}")

node_counts = {}

# Iterating over each player and their winning node
for i, x in enumerate(X_scaled):
    winning_node = som.winner(x)
    if winning_node not in node_counts:
        node_counts[winning_node] = 0
    node_counts[winning_node] += 1

# Displaying the number of players in each node
for node, count in node_counts.items():
    print(f"Node {node}: {count} players")
```

Insights

By performing Cluster Analysis on our European Football data we derived the following insights,

- The Elbow method strongly suggests n=4 as the point where adding more clusters yields diminishing returns. The Silhouette score shows a clear drop after n=2, but n=3 and n=4 still provide relatively decent scores.
- Since visuzlizing the high dimensional data directly is not feasible PCA was used to address this issue.
- By applying PCA, we ensure that the most important features are retained in the principal components. This allowed us to visualize the clusters in a lower-dimensional space while still preserving the essential structure of the data.

Association Rules

Association Rule Mining is a powerful step, especially for discovering interesting relationships or patterns between variables in our dataset. In the context of football player data, association rules can reveal hidden patterns such as "if a player has a high dribbling score, they are also likely to have a high ball control score."

- Association rule mining is commonly used for market basket analysis but can be applied to any dataset where we want to find if-then relationships between items or features.
- Rules are expressed as:
  - **Antecedent (If)**: The condition(s) (e.g., high dribbling).
  - **Consequent (Then)**: The outcome(s) (e.g., high ball control).

Key Metrics for Association Rules:

1. **Support**: How frequently the rule appears in the dataset.
2. **Confidence**: How often the consequent is true when the antecedent is true.

3. **Lift**: The strength of the rule compared to random chance.

Steps for applying association rules:

- Prepare the data by binning numerical variables and encoding categorical variables.
- Apply Apriori to identify frequent itemsets based on a support threshold.
- Generate association rules with confidence and lift metrics.
- Visualize the results to explore how attributes are linked.

On Players Table

What are the common associations between players' skill attributes?

Converting the data into Bins

```
# Defining the bins and labels for player skill levels
bin_labels = ['Low', 'Medium', 'High']

# Applying the binning to relevant player attributes
Player_Attributes_with_Names['dribbling_bin'] =
pd.cut(Player_Attributes_with_Names['dribbling'], bins=3,
labels=bin_labels)
Player_Attributes_with_Names['finishing_bin'] =
pd.cut(Player_Attributes_with_Names['finishing'], bins=3,
labels=bin_labels)
Player_Attributes_with_Names['ball_control_bin'] =
pd.cut(Player_Attributes_with_Names['ball_control'], bins=3,
labels=bin_labels)
Player_Attributes_with_Names['sprint_speed_bin'] =
pd.cut(Player_Attributes_with_Names['sprint_speed'], bins=3,
labels=bin_labels)
Player_Attributes_with_Names['strength_bin'] =
pd.cut(Player_Attributes_with_Names['strength'], bins=3,
labels=bin_labels)

# Selecting only the binned columns for association rule mining
binned_attributes = Player_Attributes_with_Names[['dribbling_bin',
'finishing_bin', 'ball_control_bin', 'sprint_speed_bin',
'strength_bin']]

# Converting the data into a format suitable for association rule
mining
df_encoded = pd.get_dummies(binned_attributes)

# Applying the Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df_encoded, min_support=0.2,
use_colnames=True)

# Generating the association rules
rules = association_rules(frequent_itemsets, metric="confidence",
```

```
min_threshold=0.6)

# Sorting the rules by confidence
rules = rules.sort_values(by='confidence', ascending=False)

# Displaying the top 10 rules
rules.head(10)
```

- Dribbling is consistently associated with multiple attributes like ball control, finishing, strength, and sprint speed, indicating that players who are good at dribbling often excel in related areas. For example, the rule "If a player has medium ball control and medium finishing, they likely have medium dribbling" has a confidence of 90.96% and a lift of 1.70, showing that this pattern is common and more likely than random occurrence.

- Players with high dribbling skills are also very likely to have high ball control (90.29% confidence and a lift of 2.35). This highlights that these two technical skills are strongly linked, as elite dribblers tend to be equally good at controlling the ball.

- Another example is the relationship between medium sprint speed and medium dribbling, which shows an 86.08% confidence. This suggests that players with average physical speed tend to also exhibit average dribbling abilities.

- The association between high dribbling and high sprint speed (74.63% confidence and a lift of 1.63) indicates that top dribblers often possess above-average speed, confirming that physical agility complements technical skill. Fast players tend to be skilled at maneuvering with the ball, making them versatile in offensive situations.

```
# Plotting the top 10 rules based on confidence
top_rules = rules.head(10)
plt.figure(figsize=(10, 6))
plt.barh(top_rules['consequents'].apply(lambda x: ', '.join(list(x))),
top_rules['confidence'], color='skyblue')
plt.xlabel('Confidence')
plt.ylabel('Rule (Consequent)')
plt.title('Top 10 Association Rules by Confidence')
plt.show()

# Creating a network graph for the association rules
G = nx.DiGraph()

for i, row in top_rules.iterrows():
    antecedent = ', '.join(list(row['antecedents']))
    consequent = ', '.join(list(row['consequents']))
    G.add_edge(antecedent, consequent, weight=row['confidence'])

# Drawing the network graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5, seed=42)
```

```
nx.draw(G, pos, with_labels=True, node_size=3000,
node_color="skyblue", font_size=10, font_weight="bold",
edge_color="gray")
plt.title('Association Rules Network')
plt.show()
```

On Teams Table

What are the common associations between team strategy attributes?

```
# Defining the bin labels for team strategy levels
bin_labels = ['Low', 'Medium', 'High']

# Applying the binning to relevant team attributes
Team_Attributes_with_Names['buildUpPlaySpeed_bin'] =
pd.cut(Team_Attributes_with_Names['buildUpPlaySpeed'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['buildUpPlayPassing_bin'] =
pd.cut(Team_Attributes_with_Names['buildUpPlayPassing'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['chanceCreationPassing_bin'] =
pd.cut(Team_Attributes_with_Names['chanceCreationPassing'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['chanceCreationCrossing_bin'] =
pd.cut(Team_Attributes_with_Names['chanceCreationCrossing'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['defencePressure_bin'] =
pd.cut(Team_Attributes_with_Names['defencePressure'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['defenceAggression_bin'] =
pd.cut(Team_Attributes_with_Names['defenceAggression'], bins=3,
labels=bin_labels)
Team_Attributes_with_Names['defenceTeamWidth_bin'] =
pd.cut(Team_Attributes_with_Names['defenceTeamWidth'], bins=3,
labels=bin_labels)

# Selecting only the binned columns for association rule mining
binned_team_attributes =
Team_Attributes_with_Names[['buildUpPlaySpeed_bin',
'buildUpPlayPassing_bin',

'chanceCreationPassing_bin', 'chanceCreationCrossing_bin',

'defencePressure_bin', 'defenceAggression_bin',

'defenceTeamWidth_bin']]

# Converting the data into one-hot encoded format
df_team_encoded = pd.get_dummies(binned_team_attributes)
```

```
# Applying Apriori to find frequent itemsets with a minimum support
threshold
frequent_itemsets = apriori(df_team_encoded, min_support=0.2,
use_colnames=True)

# Generating the association rules with confidence threshold
team_rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.6)

# Sorting the rules by confidence
team_rules = team_rules.sort_values(by='confidence', ascending=False)

# Displaying the top 10 rules
team_rules.head(10)
```

- Several rules indicate that teams with medium build-up play passing and medium chance creation strategies are strongly associated with having medium defensive aggression. For instance, the rule "If a team has medium build-up play passing and medium chance creation, they are likely to have medium defensive aggression" has a confidence of 82.55% and a lift of 1.33. This suggests that teams focusing on balanced passing and offensive creation tend to adopt a moderately aggressive defensive style.

- The lift values above 1.3 in several of these rules indicate that this relationship is significantly stronger than random chance.

- Rules involving medium chance creation passing and medium crossing show a strong relationship with medium build-up play speed. For example, the rule "If a team has medium chance creation passing and crossing, they likely adopt medium build-up play speed" has a confidence of over 80% and a lift of 1.27, meaning these offensive strategies are often used together.

- This indicates that teams with a balanced approach to passing and crossing during chance creation are more likely to focus on moderate build-up speed, balancing their offensive strategies.

- Across multiple rules, teams that show a medium level of chance creation (passing and crossing) tend to maintain similar levels of defensive aggression, indicating a balanced overall tactical strategy. The high confidence (e.g., 80.61% for passing and crossing leading to defensive aggression) shows that these tactical elements frequently occur together, suggesting teams don't typically switch drastically between offensive and defensive strategies.

```
# Ploting the top 10 rules based on confidence
top_team_rules = team_rules.head(10)
plt.figure(figsize=(10, 6))
plt.barh(top_team_rules['consequents'].apply(lambda x: ',
'.join(list(x))), top_team_rules['confidence'], color='skyblue')
```

```python
plt.xlabel('Confidence')
plt.ylabel('Rule (Consequent)')
plt.title('Top 10 Association Rules for Team Attributes by
Confidence')
plt.show()

# Creating a network graph for the top association rules
G = nx.DiGraph()

for i, row in top_team_rules.iterrows():
    antecedent = ', '.join(list(row['antecedents']))
    consequent = ', '.join(list(row['consequents']))
    G.add_edge(antecedent, consequent, weight=row['confidence'])

# Drawing the network graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.5, seed=42)
nx.draw(G, pos, with_labels=True, node_size=3000,
node_color="lightblue", font_size=10, font_weight="bold",
edge_color="gray")
plt.title('Team Attribute Association Rules Network')
plt.show()
```

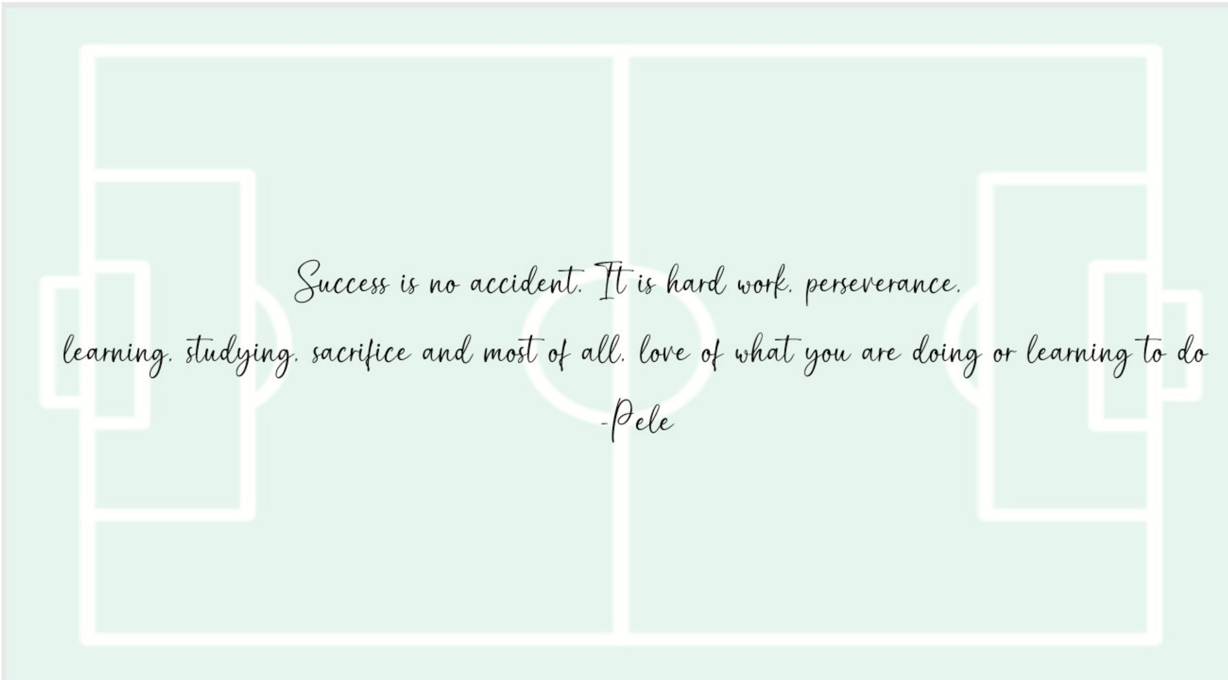
Insights

In our project, we applied association rule mining to uncover hidden patterns and relationships between player and team attributes. Using the Apriori algorithm, we were able to identify if-then patterns that show how certain skills or strategies tend to co-occur.

- Teams that adopt medium-level strategies in build-up play and chance creation (passing and crossing) often align with medium defensive aggression, highlighting a balanced tactical approach across the board.

- The association rules on Player_Attributes_with_Names provide valuable insights into how player attributes cluster together. The strong relationships between technical skills like dribbling, ball control, and finishing highlight the interdependence of these attributes, suggesting that training in one area often improves related skills. Similarly, players with balanced skill sets tend to have consistent medium-level abilities across multiple attributes.

Achievements



- Transformed the dataset into a structured format for data mining by preprocessing and engineering new features, including the binning of numerical variables to create categorical target variables for association rule mining.
- Performed hypothesis testing to assess significant differences between player groups (e.g., left-footed vs. right-footed players) and validate statistical assumptions.
- Applied various clustering techniques, such as K-Means, Gaussian Mixture Models (GMM), and Fuzzy C-Means, to group players based on performance attributes, uncovering distinct player archetypes.
- Utilized dimensionality reduction PCA for cluster visualization, making it easier to interpret high-dimensional player data and observe clustering patterns.
- Successfully applied association rule mining to uncover hidden patterns in both player and team attributes, revealing key relationships between skills and tactical strategies.
- Generated comprehensive visualizations, including bar plots, box plots, heatmaps, scatter plots, and network graphs, to effectively present the insights gained from exploratory data aanlysis, clustering, hypothesis testing, and association rule mining.

Summary of the Project

The following paragraphs are short summaries for each section in our project.

Data Preprocessing:

> Successfully handled numerical and categorical features, including scaling, encoding, and feature engineering. Applied binning for association rule mining and transformed high-dimensional data into a format suitable for clustering, hypothesis testing, and association analysis.

Hypothesis Testing:

> Conducted various hypothesis tests (e.g., t-test, Mann-Whitney U test, and Chi-square test) to evaluate significant differences between player groups and validate assumptions regarding player attributes and team strategies.

Cluster Analysis:

> Applied multiple clustering techniques, including K-Means, Gaussian Mixture Models (GMM), and Fuzzy C-Means to group players based on performance attributes. Utilized Principal Component Analysis (PCA) for dimensionality reduction, enhancing cluster visualization and interpretability.

Association Rule Mining:

> Applied the Apriori algorithm to discover significant patterns and relationships between player and team attributes, using metrics such as support, confidence, and lift to identify key associations.

References

The following is the table of References for each section of our project.

Section	Name	URL
Hypothesis Testing	Hypothesis Testing Tutorial	https://datatab.net/tutorial/hypothesis-testing
Skewness	Understanding Skewness	https://www.scribbr.com/statistics/skewness/
t-Test	t-Test Tutorial	https://datatab.net/tutorial/t-test
Mann-Whitney-u-test	Mann-Whitney U Test Tutorial	https://datatab.net/tutorial/mann-whitney-u-test
Chi-square Testing	Chi-Square Distribution Tutorial	https://datatab.net/tutorial/chi-square-distribution
Cluster Analysis	Cluster Analysis Explained	https://www.qualtrics.com/experience-management/research/cluster-analysis/
	Clustering in Data Mining	https://www.geeksforgeeks.org/clustering-in-data-mining/
	Fast Clustering Techniques (PDF)	https://www.diag.uniroma1.it/~sassano/STAGE/Fast_Clustering.pdf
	Clustering Data Mining Techniques	https://hevodata.com/learn/clustering-data-mining-techniques/
Optimal n: (Elbow, Silhouette)	Determining the Optimal Number of Clusters	https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/
K-Means Clustering	Comprehensive Guide to K-Means Clustering	https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
	Partitioning Method: K-Mean in Data Mining	https://www.geeksforgeeks.org/partitioning-method-k-mean-in-data-mining/
	Data Mining with K-Means Clustering	https://medium.com/machine-learning-and-deep-learning-alpha-quantum/data-mining-with-k-means-clustering-fd3814b86163
Gaussian Mixture Models (GMM)	Gaussian Mixture Models Documentation	https://scikit-learn.org/1.5/modules/mixture.html
	Use Cases for Gaussian Mixture Model	https://towardsdatascience.com/3-use-cases-for-gaussian-mixture-model-gmm-72951fcf8363?gi=87bec989bcb2
	Gaussian Mixture Model Explained	https://builtin.com/articles/gaussian-mixture-model
Fuzzy C-Means	Fuzzy C-Means Explained	https://www.sciencedirect.com/topics/computer-science/fuzzy-c-mean
	Understanding Fuzzy C-Means Clustering	https://www.analyticsvidhya.com/blog/2024/05/understanding-fuzzy-c-means-clustering/
Self-Organizing-Maps (SOM)	Self-Organising Maps (Kohonen Maps)	https://www.geeksforgeeks.org/self-organising-maps-kohonen-maps/
	Self-Organizing Maps on Kaggle	https://www.kaggle.com/code/abedi756/self-organizing-maps
	MiniSOM: Minimalistic and Numpy-based implementation of SOM	https://github.com/JustGlowing/minisom
	Python-SOM Implementation	https://github.com/andremsouza/python-som
Association Rules	Association Rule in Data Mining	https://www.geeksforgeeks.org/association-rule/
	Association Rules in Data Mining Explained	https://www.techtarget.com/searchbusinessanalytics/definition/association-rules-in-data-mining
	Association Rule Mining in Python	https://www.datacamp.com/tutorial/association-rule-mining-python
	Fundamentals of Associate Rule Mining	https://medium.com/image-processing-with-python/fundaments-of-associate-rule-mining-468801ec0a29

Thank You!

We express our sincere gratitude to the data providers on Kaggle. Most importantly, we extend our heartfelt thanks to our professors for inspiring and fostering the curiosity that drove us to explore this project.