POST AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY I

DEPARMENT OF PYTHON PROGRAMMING

PYTHON ASSIGNMENT

Lecturer	: Kim Ngoc Bach
Class	: D23CQCE04-B
Full name	: Tran Trung Hieu
Student ID	: B23DCCN312

Ha Noi – 2025

Table Contents

Preface

This report presents the results and methodology of the Python programming assignment for the course "Python Programming." The assignment focuses on collecting, analyzing, and visualizing football player statistics from the 2024–2025 English Premier League season. The goal of this project is to demonstrate practical skills in data scraping, processing, statistical analysis, visualization, and machine learning using Python.

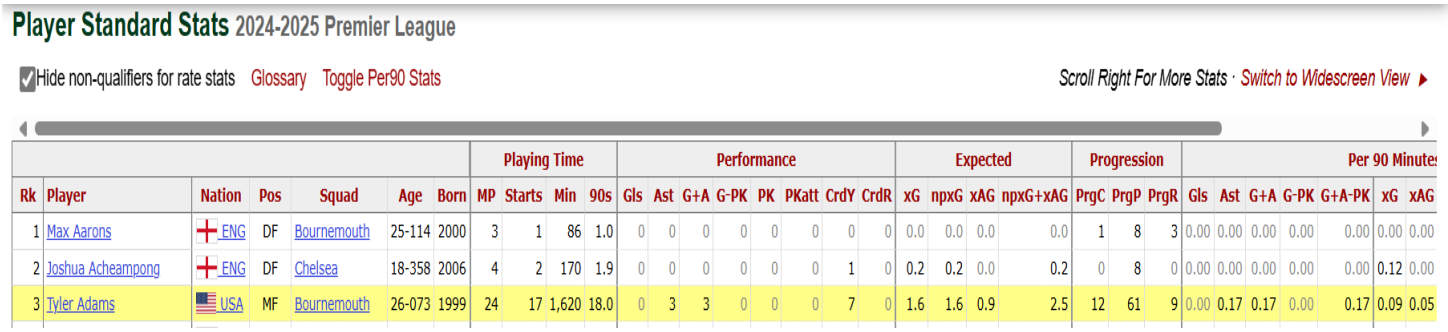
The report is organized into four main sections, each corresponding to the specific tasks outlined in the assignment. These include data collection and cleaning, descriptive and inferential statistics, player clustering using machine learning techniques, and the estimation of player transfer values based on selected features. Throughout the project, data was gathered from reputable sources such as FBref and FootballTransfers to ensure accuracy and relevance.

The tools and libraries used in the analysis include `pandas`, `numpy`, `matplotlib`, `scikit-learn`, and `selenium`, among others. The final results are saved in various output files as required, and insights are discussed in detail. This assignment has helped reinforce key programming concepts and data analysis techniques while applying them in a real-world sports analytics context.

Problem I: Collecting and Analyzing Football Player Data from FBRef

1. Problem Analysis

The goal of the task is to collect and analyze football players' statistics from the FBRef website. The data to be collected includes various metrics such as minutes played, goals scored, assists, yellow cards, and many other indicators from different sections like Shooting, Passing, Defense, Possession, Goalkeeping, etc.



Player Standard Stats 2024-2025 Premier League

Hide non-qualifiers for rate stats Glossary Toggle Per90 Stats

Scroll Right For More Stats · Switch to Widescreen View

							Playing Time				Performance								Expected				Progression			Per 90 Minutes						
Rk	Player	Nation	Pos	Squad	Age	Born	MP	Starts	Min	90s	Gls	Ast	G+A	G-PK	PK	PKatt	CrdY	CrdR	xG	npxG	xAG	npxG+xAG	PrgC	PrgP	PrgR	Gls	Ast	G+A	G-PK	G+A-PK	xG	xAG
1	Max Aarons	ENG	DF	Bournemouth	25-114	2000	3	1	86	1.0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	1	8	3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	Joshua Acheampong	ENG	DF	Chelsea	18-358	2006	4	2	170	1.9	0	0	0	0	0	0	1	0	0.2	0.2	0.0	0.2	0	8	0	0.00	0.00	0.00	0.00	0.00	0.12	0.00
3	Tyler Adams	USA	MF	Bournemouth	26-073	1999	24	17	1,620	18.0	0	3	3	0	0	0	7	0	1.6	1.6	0.9	2.5	12	61	9	0.00	0.17	0.17	0.00	0.17	0.09	0.05

2. Libraries Used

Throughout the data collection and processing, we use the following libraries:

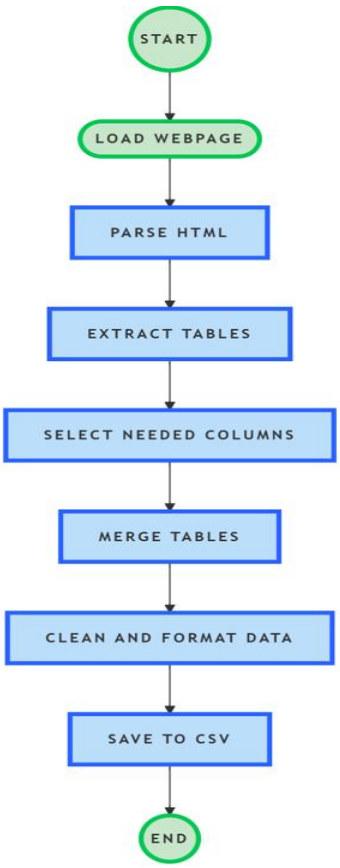
- Selenium: To automate browser interaction and collect web data.
- BeautifulSoup: To parse the HTML structure of the website and extract information.
- Pandas: To process and store the data into DataFrames.
- Undetected Chromedriver: To avoid bot detection when using Selenium.

Illustration:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
from bs4 import BeautifulSoup
import pandas as pd
import time
import undetected_chromedriver as uc
```

3. Processing Workflow

Diagram:



Step 1: Initialize the Browser and Collect Data

We use undetected_chromedriver to launch a browser and access the target web pages. Each webpage contains different data tables.

Illustration:

```
# Initialize driver
options = uc.ChromeOptions()
service = Service(ChromeDriverManager().install())
driver = uc.Chrome(service=service, options=options)
```

Browser loading FBRef page.

Step 2: Extract and Parse the Data

Using BeautifulSoup, we parse the HTML structure and locate tables by their IDs. Data is extracted from <tr> and <td> tags.

```
# Loop through all websites
for link, id in website.items():
    driver.get(link)
    time.sleep(3)
    print(f"Processing: {link}")
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    table = soup.find('table', id=id)
```

Step 3: Clean and Transform the Data

We perform steps such as converting data from strings to numbers, removing missing values, and keeping only players with more than 90 minutes played and sort players by first name.

```
# Process 'minutes played' data
df_stat_standard = tables['stats_standard']
df_stat_standard['Min'] = pd.to_numeric(df_stat_standard['Min'].str.replace(',', ''), errors='coerce')

# Filter players who played more than 90 minutes
df_stat_standard = df_stat_standard[df_stat_standard['Min'] > 90]
df_stat_standard = df_stat_standard.sort_values(by='Player')  # Sort players by name
```

Step 4: Select Necessary Columns

Since each statistics table (such as Shooting, Passing, Defense, etc.) contains many columns, we select only the essential columns needed for analysis by specifying their column indices.

After selecting the necessary columns, we append the resulting DataFrame into the list bang[] for later merging.

Example: Selecting important columns in the 'misc' table:

```
idx_drop_misc = [10, 11, 12, 13, 19, 20, 21, 22]
cols = [0, 1, 2, 3] + idx_drop_misc
tables['stats_misc'] = tables['stats_misc'].iloc[:, cols]
bang.append(tables['stats_misc'])
```

- [0,1,2,3] corresponds to Player, Nation, Squad, Position columns — always essential.
- idx_drop_misc adds extra selected columns for this specific table.
- This step ensures that only necessary columns are extracted from each data table.
- Finally, the processed table is appended into bang[], which stores all tables for future merging.

Step 5: Merge Different Data Tables

After collecting and cleaning the data, we merge tables from different sections like "Passing," "Shooting," "Defense," etc., into one combined table.

```
# Merge all tables based on Player, Nation, Squad, and Pos
for table in bang[1:]:
    df_stat_standard = pd.merge(df_stat_standard, table, on=["Player", "Nation", "Squad", "Pos"], how='left')
```

Step 6: Final Processing and Save the Results

Before saving the final CSV file, we perform several important post-processing steps:

Fill Missing Values

We replace empty strings and missing values (NaN) with "N/a" to ensure consistency:

```
# Replace missing values with "N/a"
df_stat_standard = df_stat_standard.replace("", "N/a").fillna("N/a")
```

Convert Nation Names

We convert the full nation names into uppercase abbreviations using the convert_nation() function:

```
# Convert nation names to abbreviations
df_stat_standard['Nation'] = df_stat_standard['Nation'].apply(convert_nation)
```

Remove Duplicate Players

To avoid redundancy, we remove duplicate player entries, keeping only the first occurrence:

```
# Remove duplicate players (keep first occurrence)
df_stat_standard = df_stat_standard.drop_duplicates(subset=['Player', 'Squad'])
```

Rename Columns

We set the final standardized column names based on a predefined header list:

```
header = [
    "Player", "Nation", "Pos", "Team", "Age", "MP", "Starts", "Min", "Gls", "Ast", "CrdY", "CrdR", "xG", "xAG",
    "progression_prgc", "progression_prgp", "progression_prgr", "per90_gls", "per90_ast", "per90_xg", "per90_xag",
    "goalkeeping_performance_ga90", "goalkeeping_performance_savepct", "goalkeeping_performance_cspct", "goalkeeping_penalties_savepct",
    "shooting_standard_sotpct", "shooting_standard_sot_per90", "shooting_standard_g_sh", "shooting_standard_dist",
    "passing_total_cmp", "passing_total_cmppct", "passing_total_totdist", "passing_short_cmppct", "passing_medium_cmppct",
    "passing_long_cmppct", "passing_expected_kp", "passing_expected_1_3", "passing_expected_ppa", "passing_expected_crspa",
    "passing_expected_prgp", "creation_sca_sca", "creation_sca_sca90", "creation_gca_gca", "creation_gca_gca90",
    "defense_tackles_tkl", "defense_tackles_tklw", "defense_challenges_att", "defense_challenges_lost",
    "defense_blocks_blocks", "defense_blocks_sh", "defense_blocks_pass", "defense_blocks_int",
    "possession_touches_touches", "possession_touches_def_pen", "possession_touches_def_3rd",
    "possession_touches_mid_3rd", "possession_touches_att_3rd", "possession_touches_att_pen", "possession_takeons_att",
    "possession_takeons_succpct", "possession_takeons_tkldpct", "possession_carries_carries",
    "possession_carries_prgdist", "possession_carries_prgc", "possession_carries_1_3",
    "possession_carries_cpa", "possession_carries_mis", "possession_carries_dis", "possession_receiving_rec",
    "possession_receiving_prgr", "misc_performance_fls", "misc_performance_fld", "misc_performance_off",
    "misc_performance_crs", "misc_performance_recov", "misc_aerial_won", "misc_aerial_lost", "misc_aerial_wonpct"
]

# Swap "Team" and "Pos" columns
df_stat_standard.columns = header
```

Swap "Team" and "Pos" Columns

For better data organization, we switch the positions of the "Team" and "Pos" columns:

```
# Swap "Team" and "Pos" columns
df_stat_standard.columns = header
cols = list(df_stat_standard.columns)
i, j = cols.index('Team'), cols.index('Pos')
cols[i], cols[j] = cols[j], cols[i]
df_stat_standard = df_stat_standard[cols]
```

Save to CSV

Finally, we export the processed DataFrame into a CSV file:

```
# Save the final DataFrame to CSV
current_dir = Path(__file__).parent
file_path = current_dir / 'results.csv'
df_stat_standard.to_csv(file_path, index=False, encoding='utf-8-sig')
```

4. Results

The final data is stored in a CSV file. Metrics such as goals, assists, minutes played, and others from multiple sections are combined, providing a comprehensive view of each player's performance.

Final CSV Example:

1	Player	Nation	Team	Pos	Age	MP	Starts	Min	Gls	Ast	CrdY	CrdR	xG	xAG	progression_prgc	progression_prgp	progression_prgr
2	Aaron Cresswell	ENG	West Ham	DF	35-134	14	7	589	0	0	2	0	0.1	1.1	4	24	2
3	Aaron Ramsdale	ENG	Southampton	GK	26-349	26	26	2340	0	0	2	0	0.0	0.0	0	0	0
4	Aaron Wan-Bissaka	ENG	West Ham	DF	27-153	32	31	2794	2	2	1	0	1.2	2.9	98	125	139
5	Abdoulaye Doucouré	MLI	Everton	MF	32-117	30	29	2425	3	1	5	1	3.9	2.3	40	78	91
6	Abdukodir Khusanov	UZB	Manchester City	DF	21-058	6	6	503	0	0	1	0	0.0	0.1	1	25	2
7	Abdul Fatawu Issahaku	GHA	Leicester City	FW	21-051	11	6	579	0	2	0	0	0.4	1.6	42	17	60
8	Adam Armstrong	ENG	Southampton	FW,MF	28-077	20	15	1248	2	2	4	0	3.3	1.2	25	21	79
9	Adam Lallana	ENG	Southampton	MF	36-353	14	5	361	0	2	4	0	0.2	0.9	6	24	10
10	Adam Smith	ENG	Bournemouth	DF	33-364	22	17	1409	0	0	6	0	0.7	0.3	12	40	31
11	Adam Webster	ENG	Brighton	DF	30-114	11	8	617	0	0	0	0	0.0	0.5	7	40	2
12	Adam Wharton	ENG	Crystal Palace	MF	20-330	19	15	1258	0	2	2	0	0.3	3.0	14	105	10

Conclusion:

This project successfully collected statistical data from FBRef, cleaned and processed it, and created a combined table summarizing various player metrics. Each step from data collection, cleaning, table merging, to saving results was executed effectively. The objective of the task was achieved, and the final CSV file is ready for further analysis.

Problem II :

II.1: Analyzing Football Player Statistics (Top and Bottom Performers)

1.Problem Analysis

The goal of this task is to analyze football player data from a CSV file (results.csv) and identify the top and bottom 3 players in various statistical categories. The statistics include metrics such as goals, assists, minutes played, and others. We will process the data to compute and extract the top and bottom performers for each metric.

2.Libraries Used

The following libraries are used to achieve the task:

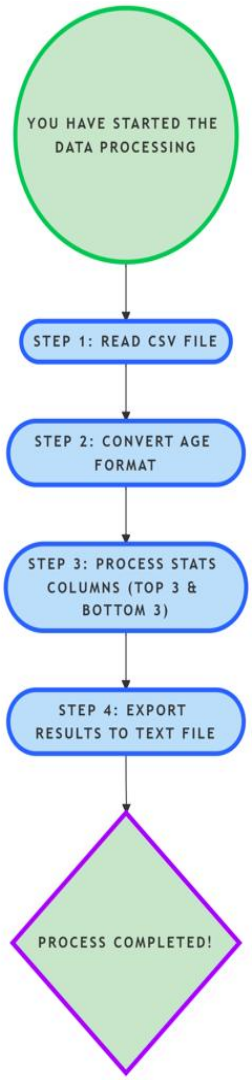
- Pandas: For reading, manipulating, and processing the data stored in the CSV file.
- NumPy: For numerical operations and handling data.

Illustration:

```
import pandas as pd
from pathlib import Path
```

3. Processing Workflow

Diagram:



Step 1: Reading the CSV File:

First, the data is read from the CSV file using pandas' read_csv() function.

Illustration:

1	Player	Nation	Team	Pos	Age	MP	Starts	Min	Gls	Ast	CrdY	CrdR	xG	xAG	progression_prgc	progression_prgp	progression_prgr
2	Aaron Cresswell	ENG	West Ham	DF	35-134	14	7	589	0	0	2	0	0.1	1.1	4	24	2
3	Aaron Ramsdale	ENG	Southampton	GK	26-349	26	26	2340	0	0	2	0	0.0	0.0	0	0	0
4	Aaron Wan-Bissaka	ENG	West Ham	DF	27-153	32	31	2794	2	2	1	0	1.2	2.9	98	125	139
5	Abdoulaye Doucouré	MLI	Everton	MF	32-117	30	29	2425	3	1	5	1	3.9	2.3	40	78	91
6	Abdukodir Khusanov	UZB	Manchester City	DF	21-058	6	6	503	0	0	1	0	0.0	0.1	1	25	2

Step 2: Convert Age Format

We convert the player age from the "year-days" format into a decimal format using the convert_age() function. This function splits the age into years and

```python
# Function to convert age string format 'year-days' into a decimal number
def convert_age(age_str):
    if isinstance(age_str, str) and '-' in age_str:
        try:
            year, days = map(int, age_str.split('-'))
            return round((year + days / 365), 2)
        except:
            return None
    return None  # Return None if not a string or wrong format
```

days and converts them into a decimal.

Step 3: Data Processing

After converting the age, we apply the process of selecting top and bottom performers based on each statistical column (such as Goals, Assists, etc.). For each statistic:

- Convert the column to numeric values.
- Drop rows where the values are missing (NaN).
- Identify the top 3 players with the highest values in the statistic.
- Identify the bottom 3 players with the lowest values in the statistic.

Step 4: Export Results to Text File

The results for each statistical category (top 3 and bottom 3) are saved into a text file.

Illustration:

```
# Export results to text file
current_dir = Path(__file__).parent
file_path = current_dir / 'top_3.txt'
with open(file_path, 'w', encoding='utf-8') as f:
    for item in a:
        if isinstance(item, str):
            f.write(item + '\n')
        else:
            f.write(item.to_string(index=False) + '\n\n')  # Print DataFrame nicely
```

4.Results

The final output is stored in a text file (top_3.txt). For each statistical metric, we output the top 3 players with the highest values and the bottom 3 players with the lowest values.

Final Text File Example:

```
----------Top 3 players with highest Age: ----------
          Player Nation     Team   Pos   Age
Łukasz Fabiański    POL West Ham    GK 40.03
   Ashley Young    ENG  Everton DF,FW 39.80
   James Milner    ENG Brighton    MF 39.31

----------Bottom 3 players with lowest Age: ----------
       Player Nation           Team   Pos   Age
  Mikey Moore    ENG      Tottenham FW,MF 17.71
Ethan Nwaneri    ENG        Arsenal FW,MF 18.10
  Harry Amass    ENG Manchester Utd    DF 18.12

----------Top 3 players with highest MP: ----------
          Player Nation          Team   Pos  MP
     Alex Iwobi    NGA        Fulham FW,MF  34
     Bernd Leno    GER        Fulham    GK  34
Bruno Guimarães    BRA Newcastle Utd    MF  34
```

Conclusion:

This script successfully analyzes the football player data, identifying the top 3 and bottom 3 performers for each statistical category. The output is saved in a text file for easy review. The processing steps, including data reading, cleaning, conversion, and result extraction, were carried out efficiently, providing valuable insights into the players' performances.

II.2: Summary Statistics of Football Player Metrics by Team

1. Problem Analysis

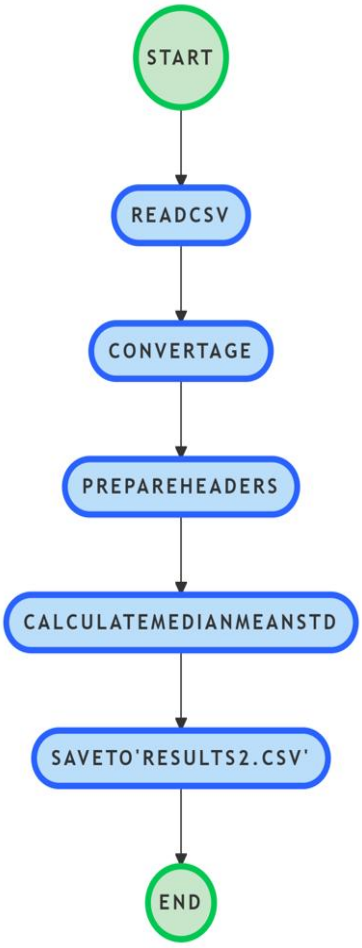
The objective of this task is to compute summary statistics — median, mean, and standard deviation — for each performance metric (e.g., goals, assists, minutes played, etc.) across all players and for each team. The summarized results are saved to a file named results2.csv.

2. Libraries Used

- Pandas: For reading and processing tabular data.
- NumPy: For statistical computations.

3. Processing Workflow

Diagram:



Step 1: Read the CSV File

The original data (results.csv) is loaded using pandas.read_csv().

```
# Read the CSV file and apply age conversion
current_dir = Path(__file__).parent
file_path = current_dir / 'results.csv'
df = pd.read_csv(file_path)
```

Step 2: Convert Age Format

The player age is originally in the format "year-days" (e.g., "22-135"). A helper function convert_age() is applied to transform this into decimal format (e.g., 22.37).

```
df['Age'] = df['Age'].apply(convert_age)
```

Step 3: Prepare Output Columns

A header list is created with labels for Median, Mean, and Std for every numeric metric starting from the 5th column (ignoring player metadata).

```
# Prepare the header for the result table
a = ['']
for head in headers[4:]:
    a.append(f'Median of {head}')
    a.append(f'Mean of {head}')
    a.append(f'Std of {head}')
```

Step 4: Calculate Statistics

For each team (and an overall "all" category):

- Filter the data for the team.

- Convert each metric column to numeric.
- Calculate:
    - Median using .median()
    - Mean using .mean()
    - Standard deviation using .std()
- Append results to a master list result_rows.

Step 5: Save to CSV

All statistics are combined into a single DataFrame and saved to results2.csv using UTF-8 encoding.

```
# (Optional) Save the result to a CSV file
current_dir = Path(__file__).parent
file_path = current_dir / 'results2.csv'
summary_df.to_csv(file_path, index=True, encoding='utf-8-sig')  # Save to CSV with UTF-8 encoding
```

4. Results

The resulting file results2.csv contains rows for each team (and one for all players) with the median, mean, and standard deviation for every metric. This enables easy comparison of performance across teams and identification of trends.

Example Output Structure (Partial):

			Median of Age	Mean of Age	Std of Age	Median of MP	Mean of MP	Std of MP	Median of Starts	Mean of Starts	Std of Starts	Median of Min	Mean of Min
1	0	all	26.56	26.89	4.23	22.0	20.67	9.64	14.0	15.16	10.64	1335.0	1359.86
2	1	Arsenal	26.86	26.48	3.82	22.5	22.59	7.93	16.0	17.0	10.16	1427.5	1519.45
3	2	Aston Villa	27.62	27.12	4.04	20.0	18.86	9.97	9.5	13.36	11.32	969.0	1200.0
4	3	Bournemouth	25.98	26.45	3.84	25.0	21.61	9.84	17.0	16.22	11.33	1580.0	1451.74
5	4	Brentford	24.82	26.18	3.91	26.0	22.24	10.84	21.0	17.29	12.63	1913.0	1547.67
6	5	Brighton	24.5	26.02	5.17	20.0	18.96	9.76	9.0	13.36	9.87	895.5	1198.46
7	6	Chelsea	24.3	24.18	2.38	17.5	19.15	10.88	11.5	14.38	11.4	1046.5	1291.42
8	7	Crystal Palace	27.17	27.15	3.14	29.0	23.38	10.21	18.0	17.71	12.47	1562.0	1585.9
9	8	Everton	26.52	27.96	5.03	23.5	21.73	9.17	14.5	16.95	10.56	1281.0	1520.27

Conclusion

This script automates the generation of key statistical indicators for football players by team. It supports effective performance benchmarking at both team and overall levels, with outputs saved in a clean, structured format.

II.3.1: Visualizing Football Player Statistics Using Histograms

1. Problem Analysis

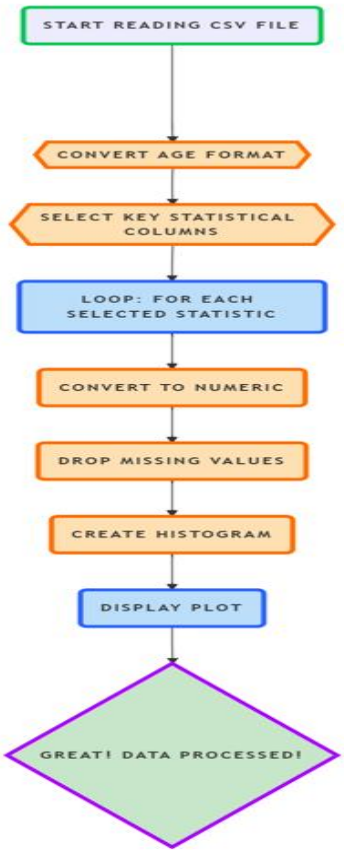
The goal of this task is to visualize football player statistics by generating histograms for various performance metrics. The data is stored in a CSV file (results.csv), and we aim to better understand the distribution of key statistics such as goals, shooting accuracy, and defensive challenges.

Diagram:

2. Libraries Used

- Pandas: For reading and manipulating the dataset.
- Matplotlib: For plotting histograms to visualize data distribution.

3. Processing Workflow



Step 1: Reading the CSV File

The dataset is read from a CSV file using pandas.read_csv().

Example:

```
# Load the data from the specified file path
current_dir = Path(__file__).parent
file_path = current_dir / 'results2.csv'
df = pd.read_csv(file_path)
```

Step 2: Age Conversion

The players' ages, initially in a year-days format, are converted into decimal years using a custom convert_age() function.

Example:

df['Age'] = df['Age'].apply(convert_age)

Step 3: Data Preparation

A list of key statistical columns is selected for visualization, such as:

- Goals scored (Gls)
- Shooting accuracy (shooting_standard_sotpct)
- Shots on target per 90 minutes (shooting_standard_sot_per90)
- Defensive tackles won (defense_tackles_tklw)
- Defensive challenges attempted (defense_challenges_att)

- Defensive challenges lost (defense_challenges_lost)

```
# Define the list of statistical columns we want to create histograms for
headers = ["Gls", "shooting_standard_sotpct", "shooting_standard_sot_per90",
           "defense_tackles_tklw", "defense_challenges_att", "defense_challenges_lost"
           ]
```

Step 4: Generating Histograms

For each selected statistic:

- Convert the column values to numeric.
- Drop missing values.
- Create and display a histogram showing the frequency distribution.

Example:

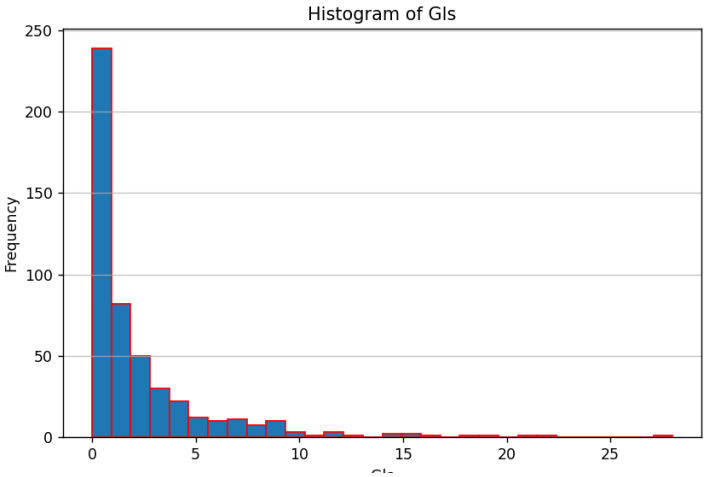
```
# Create the histogram plot
plt.figure(figsize=(10, 6))  # Set the figure size
plt.hist(df_copy[head], bins=30, edgecolor='red')  # Plot the histogram with 30 bins and r

# Set the title and labels
plt.title(f'Histogram of {head}')
plt.xlabel(head)
plt.ylabel('Frequency')

# Add grid lines for better readability
plt.grid(axis='y', alpha=0.75)

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

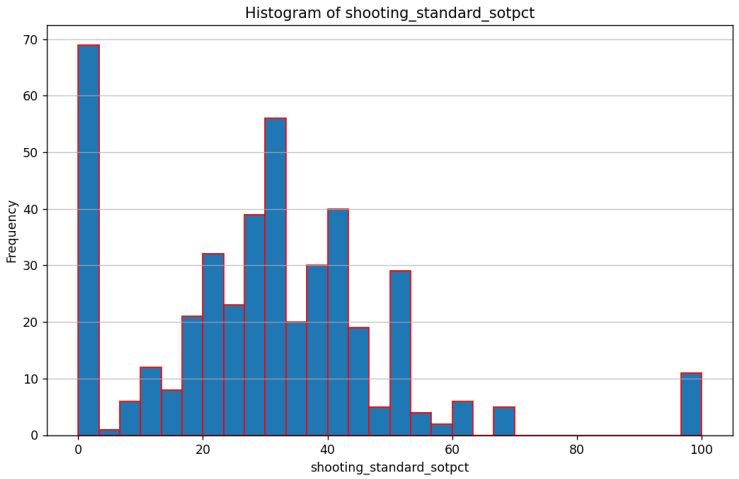
4. Results

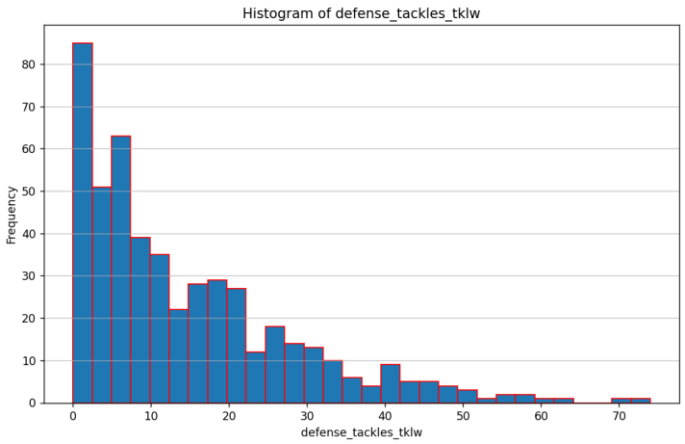
- For each selected statistic, a histogram is generated, providing a clear visualization of its distribution across players.

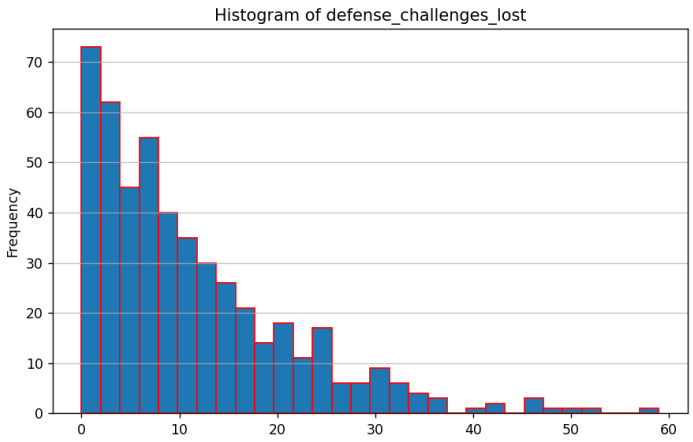
- Outliers and trends (e.g., most players scoring fewer goals, defensive success rates) are easily observed through the plotted graphs.

Sample Figures:









Conclusion

This script successfully visualized football player statistics, helping to identify trends, outliers, and common performance levels. Histograms provide an intuitive way to analyze the overall distribution and variability among players for various performance metrics.

II.3.2: Visualizing Football Team Statistics Using Histograms

1. Problem Analysis

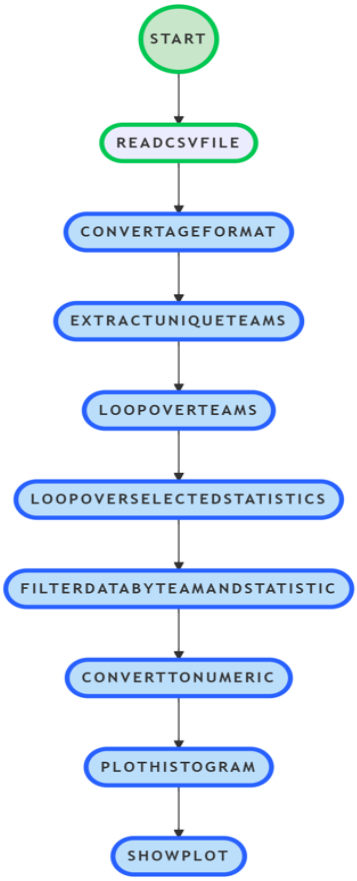
The objective of this task is to visualize football team statistics by creating histograms for key performance metrics. Each team's data is analyzed separately to observe internal distributions and performance patterns.

2. Libraries Used

- Pandas: For reading the dataset and data preprocessing.
- Matplotlib: For generating histograms for each team and each statistic.

3. Processing Workflow

Diagram:



Step 1: Read the Data

Load the player data from the CSV file results.csv.

Example:

```
# Load the data from the specified file path
current_dir = Path(__file__).parent
file_path = current_dir / 'results2.csv'
df = pd.read_csv(file_path)
```

Step 2: Age Conversion

Convert player ages from year-days format into decimal years.

Example:

df['Age'] = df['Age'].apply(convert_age)

Step 3: Extract Unique Teams

Identify all unique team names for grouping the players.

teams = sorted(list(set(df['Team'])))

Step 4: Plot Histograms

For each team and each selected statistic:

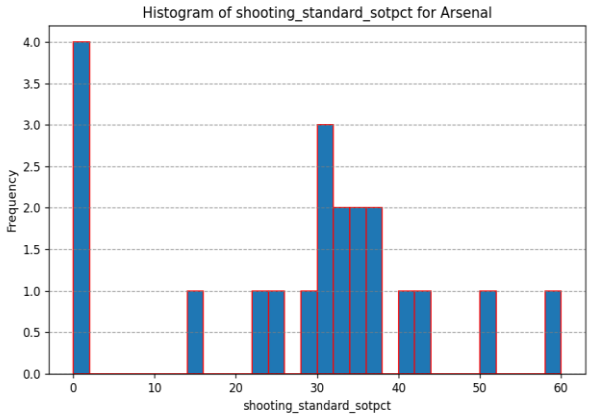
- Filter players by team.
- Convert the statistic to numeric values.
- Plot a histogram to visualize the data distribution.

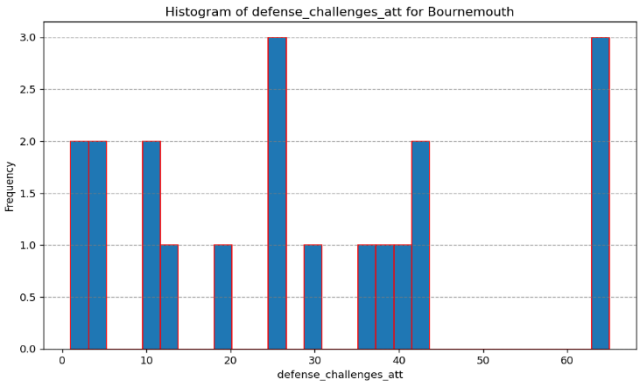
5. Results

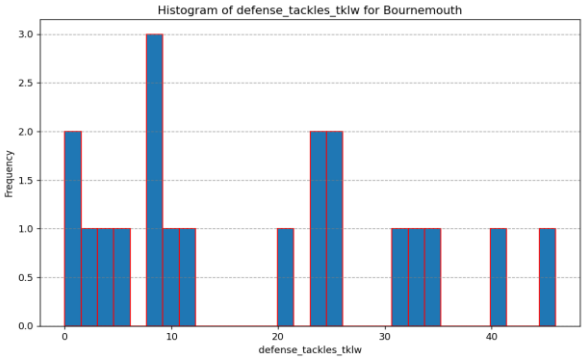
- Each team has a set of histograms for the following statistics:
    - Goals (Gls)
    - Shooting accuracy (shooting_standard_sotpct)
    - Shots on target per 90 minutes (shooting_standard_sot_per90)
    - Tackles won (defense_tackles_tklw)
    - Challenges attempted (defense_challenges_att)
    - Challenges lost (defense_challenges_lost)
    - Blocks made (defense_blocks_blocks)
- Histograms show how each team's players are distributed across these metrics.

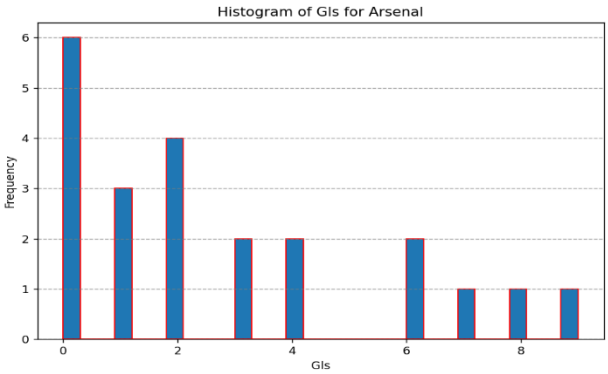
```
# Create the histogram plot for the current column and team
plt.figure(figsize=(10, 6))  # Set the figure size
plt.hist(df_copy[head], bins=30, edgecolor='red')  # Plot the histogram with 30 bins and red edges

# Set the title and labels for the plot
plt.title(f'Histogram of {head} for {team}')
plt.xlabel(head)
plt.ylabel('Frequency')

# Add grid lines for better readability
plt.grid(axis='y', alpha=0.75, linestyle='--', color='gray')

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

Example Figures:









Conclusion

This task successfully visualized football statistics by team. Histograms allow easy comparison of player distributions within each team and highlight areas where teams excel or underperform.

II.4: Identifying the Best Performing Team in the 2024-2025 Premier League

1. Problem Analysis

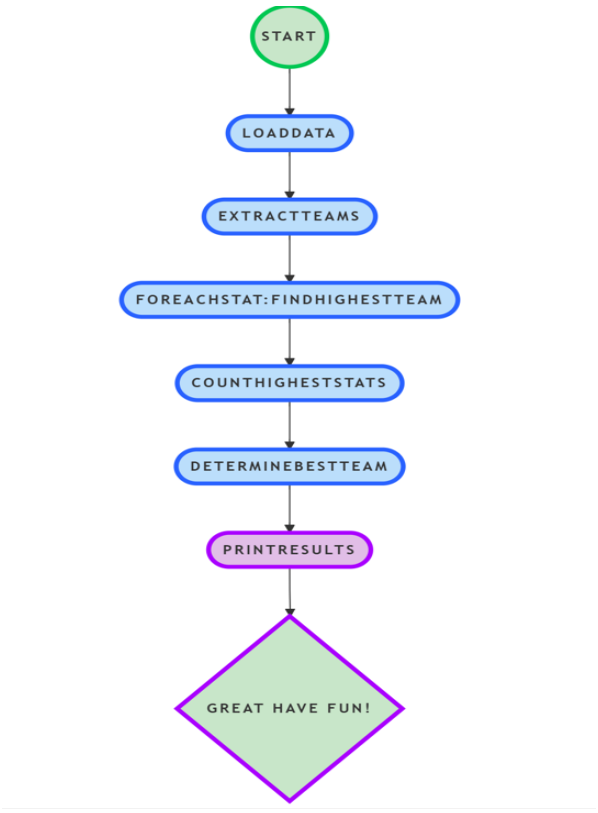
The objective of this task is to analyze the football statistics from the results2.csv file and identify which team has the highest scores for each statistic. We are particularly interested in identifying the team that performs the best overall in the 2024-2025 Premier League season based on these statistics.

Diagram:

2. Libraries Used

- Pandas: Used for reading and processing the CSV data.

3. Processing Workflow



Step 1: Load the Data

The results2.csv file is loaded into a pandas DataFrame, which contains various statistics for teams across different categories.

```
# Load the CSV file into a DataFrame
current_dir = Path(__file__).parent
file_path = current_dir / 'results2.csv'
df = pd.read_csv(file_path)
```

Step 2: Extract Teams

A set of teams is extracted from the second column of the DataFrame (excluding the header).

```
teams = set(df.iloc[1:, 1])
```

Step 3: Identify Highest Stat for Each Metric

We extract the relevant statistical categories from the DataFrame, iterating over each and identifying the team with the highest score.

```
headers = list(df.columns)[3::3]
```

Step 4: Count the Number of Times Each Team Has the Highest Score

A dictionary danh_sach_highest is used to track the number of times each team has the highest score for any given statistic. The team with the most "highest" scores across all statistics is considered the best performer.

```
danh_sach_highest[team] += 1
```

Step 5: Determine the Top Team

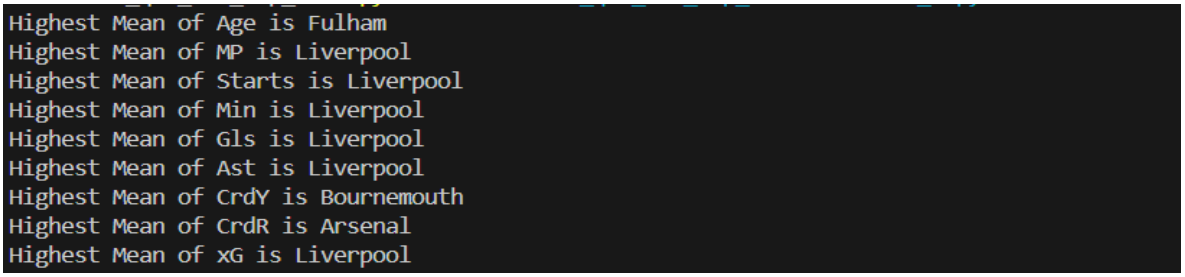
After analyzing all statistics, the team with the most "highest" statistics is identified as the best-performing team.

```
team_win = max(danh_sach_highest, key=danh_sach_highest.get)
```

4. Results

After running the code, the team with the highest values for each statistic is printed. The team that appears most frequently as the highest performer is deemed the best overall performer.

For example, the output may look like:



Finally, the team with the most "Highest" scores across all statistics is identified as the best performing team in the 2024-2025 season.

Final Output:

- The best-performing team is LiverPool , based on the highest number of top statistics.

Conclusion

This task efficiently identifies the team with the best performance by analyzing the highest scores for various football statistics. The team that dominates the most categories can be considered as the strongest contender in the current season.

Problem III :

III.1: K-means Clustering for Player Statistics

1. Problem Analysis

In this task, we aim to classify players into groups based on their statistics using the K-means algorithm. By doing so, we will determine how many clusters (or groups) are optimal for categorizing players. Additionally, we will provide insights into the number of clusters that best represent the players' data.

2. Approach

We performed clustering using the K-means algorithm and analyzed the results using two main techniques:

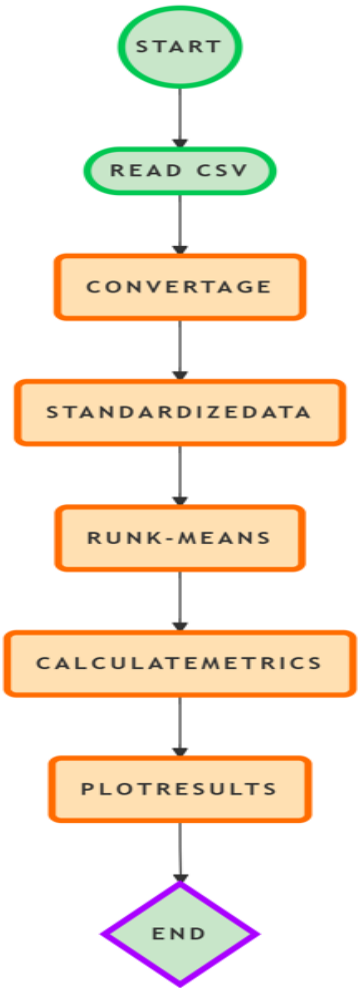
- Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters. A higher silhouette score indicates that the data points are well-clustered.
- Elbow Method: Helps in determining the optimal number of clusters by plotting the inertia (sum of squared distances of samples to their centroids) for different values of k. The "elbow" point indicates the optimal k.

3. Preprocessing Steps

- Data Standardization: The data was standardized using StandardScaler to bring all statistics to a similar scale, which is crucial for clustering algorithms.

- Age Conversion: The age column was converted from a string format (e.g., "25-150") to decimal years using a custom function.

4. Code Workflow



Step 1: Data Preparation

- The dataset was loaded from a CSV file, and the Age column was processed to standardize it.
- Non-numeric values like 'N/a' were replaced with 0 for consistency.

Step 2: Clustering

- The relevant statistics for clustering were extracted from the dataset. The columns were standardized to ensure each statistic contributed equally to the clustering.

Step 3: K-means Clustering

- Inertia was calculated for different values of k (number of clusters) to apply the Elbow Method.
- Silhouette Score was calculated for different k values to evaluate the quality of the clusters.

Step 4: Visualization

Two plots were generated:

1. Silhouette Scores for k-values from 2 to 74.
2. Inertia (Elbow Method) for k-values from 2 to 74.

```
# Create a figure with two subplots for visualizing results
fig, ax = plt.subplots(1, 2, figsize=(14, 5))

# Plot silhouette scores
ax[0].plot(range(2, 75), sil_score, marker='o')  # Plot silhouette scores for k = 2 to 74
ax[0].set_title('Silhouette Score')  # Set the title of the plot
ax[0].set_xlabel('Number of clusters')  # Label for the x-axis
ax[0].set_ylabel('Silhouette Score')  # Label for the y-axis
ax[0].axvline(x=8, color='red', linestyle='--')  # Add a vertical line at k=8 (chosen for analysis)

# Plot inertia (Elbow Method)
ax[1].plot(range(2, 75), inertia, marker='o')  # Plot inertia values for k = 2 to 74
ax[1].set_title('Elbow Method')  # Set the title of the plot
ax[1].set_xlabel('Number of clusters')  # Label for the x-axis
ax[1].set_ylabel('Elbow')  # Label for the y-axis
ax[1].axvline(x=8, color='red', linestyle='--')  # Add a vertical line at k=8 (suggested by the elbow method)

# Set the overall title for the plots
plt.suptitle('KMeans Clustering Analysis')

# Adjust the layout to ensure no overlapping of plots
plt.tight_layout()

# Display the plots
plt.show()
```

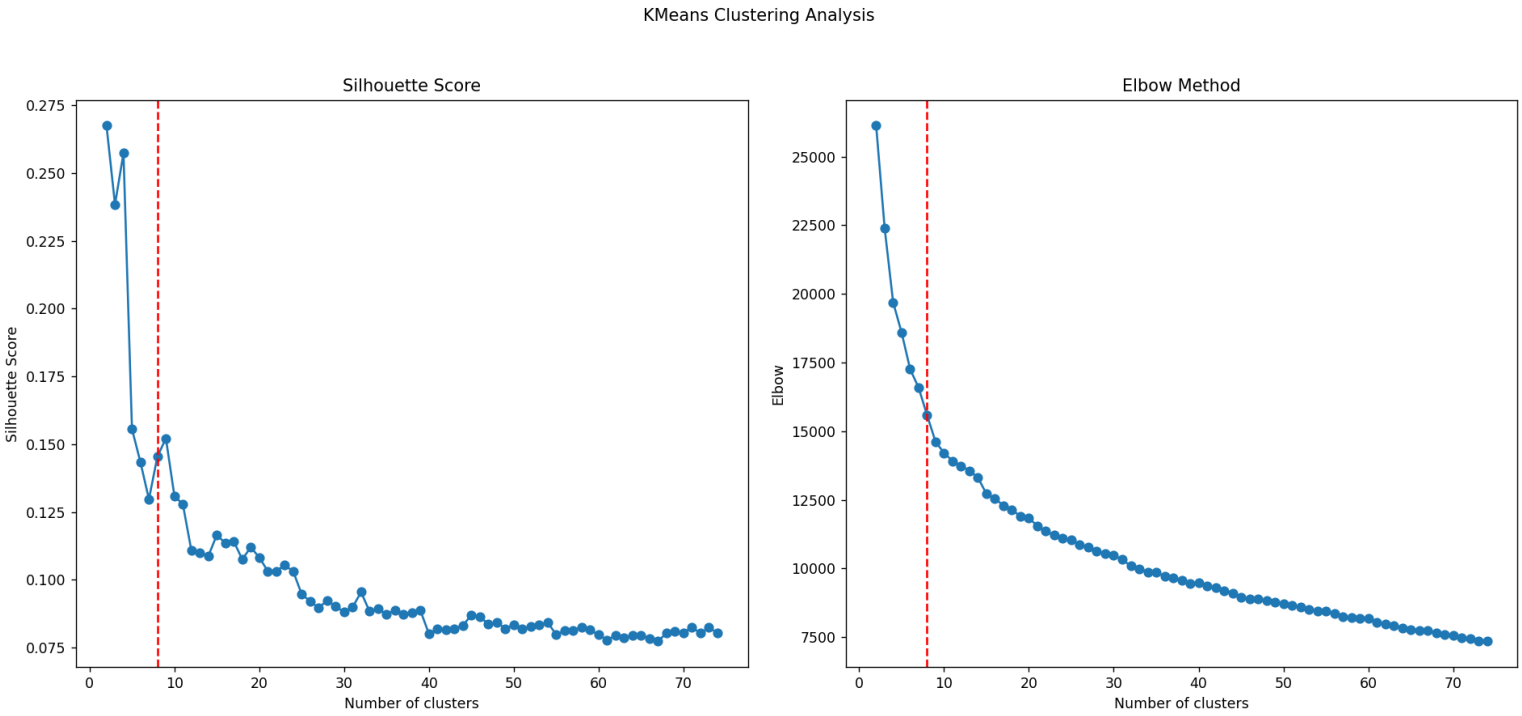
5. Results

- The Silhouette Score plot helps determine how well-defined the clusters are for different values of k. Higher values indicate better-defined clusters.

  The Elbow Method plot shows a clear "elbow" at k=8, suggesting that 8 clusters is a good choice for the optimal number of clusters.

Key Observations:

- The Silhouette Score reaches its highest around k=8, indicating the best cluster quality.
- The Elbow Method also points to k=8 as the optimal number of clusters based on the inertia plot.



Conclusion

Based on both the Silhouette Score and Elbow Method, 8 clusters is the optimal number of clusters for classifying the players based on their statistics. This suggests that the players can be grouped into 8 different categories that represent distinct performance levels or characteristics.

III.2: Dimensionality Reduction and Clustering using PCA and K-means

1. Problem Overview

The objective of this task is to apply Principal Component Analysis (PCA) for dimensionality reduction and subsequently perform K-means clustering on the reduced data. The goal is to classify players into clusters based on their statistics and visualize the clustering results in a 2D space.

2. Approach

Step 1: Data Preprocessing

- Age Conversion: The age column, initially in a "year-days" format, was converted to a decimal year format using a custom function.
- Handling Missing Data: Non-numeric values such as 'N/a' were replaced with 0 to ensure consistency in the dataset.
  Standardization: Since clustering algorithms are sensitive to the scale of the data, all relevant player statistics were standardized using StandardScaler to ensure equal contribution from each feature.

Step 2: Dimensionality Reduction using PCA

- PCA was applied to reduce the dataset to 2 dimensions (PC1 and PC2) for better visualization of the clustering results. This allows us to plot the players in a 2D space, making it easier to see how the clustering algorithm groups similar players.

Step 3: Clustering using K-means

- The K-means clustering algorithm was applied for various values of clusters (7 to 12) to observe how the players are grouped.
- For each clustering configuration, the data points were colored based on their assigned cluster, and the results were visualized on a 2D scatter plot.

3. Results

PCA Visualization

The scatter plots below show how the players are distributed in the 2D space formed by the first two principal components (PC1 and PC2).

- X-axis: Principal Component 1 (PC1)
- Y-axis: Principal Component 2 (PC2)
- Each point represents a player, and the color corresponds to the assigned cluster.

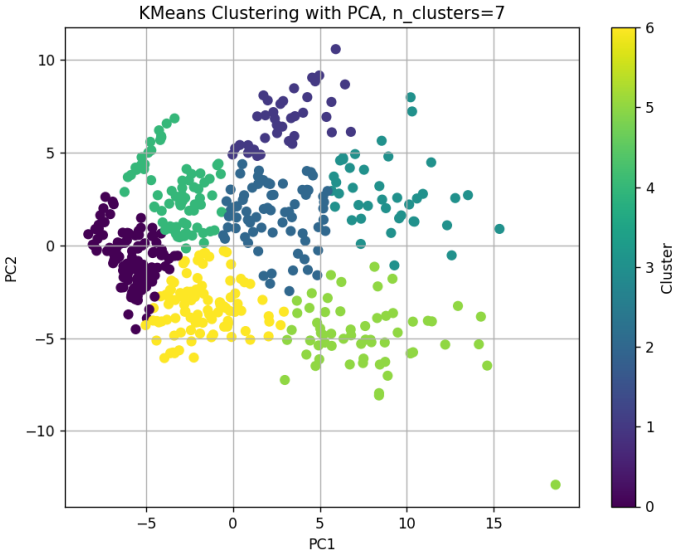
Clustering Results:

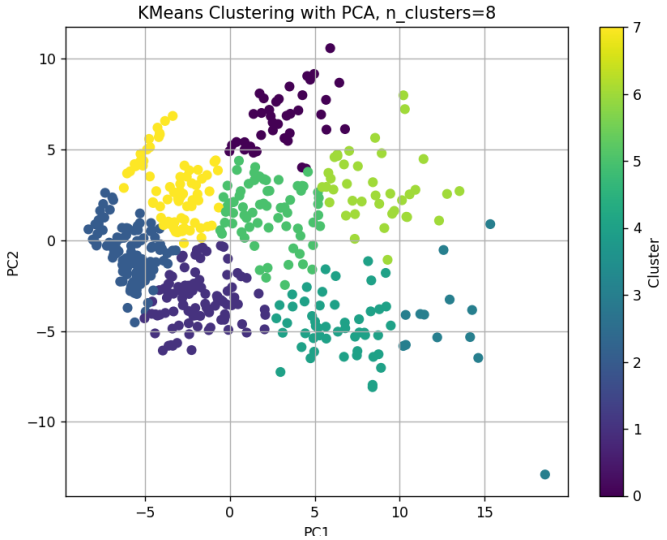
- The plots were generated for different numbers of clusters: 7, 8, 9, 10, 11, and 12.
- As the number of clusters increases, the separation between groups becomes more refined. However, the choice of the optimal number of clusters depends on the structure of the data.

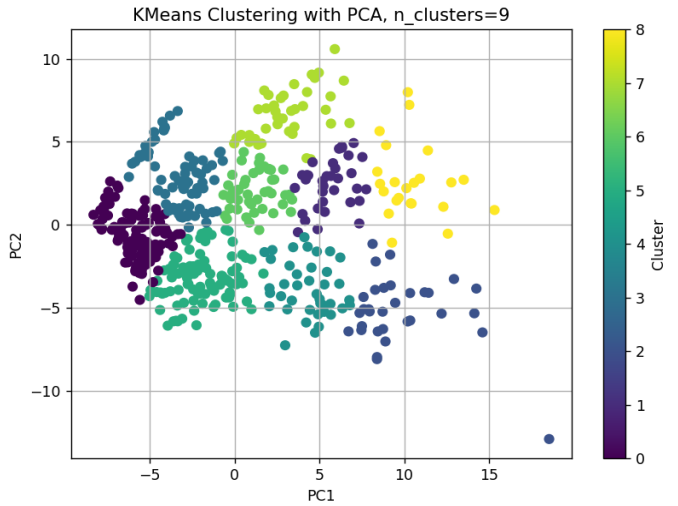
Observations:

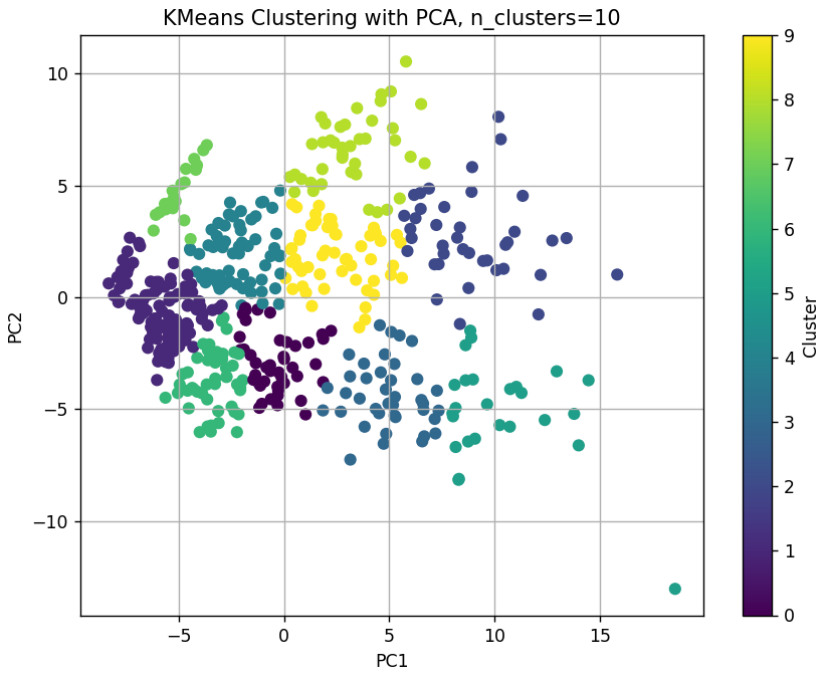
- Cluster separation: As more clusters are added, the data points in the scatter plots become more dispersed and separate, suggesting finer categorization of players based on their statistics.

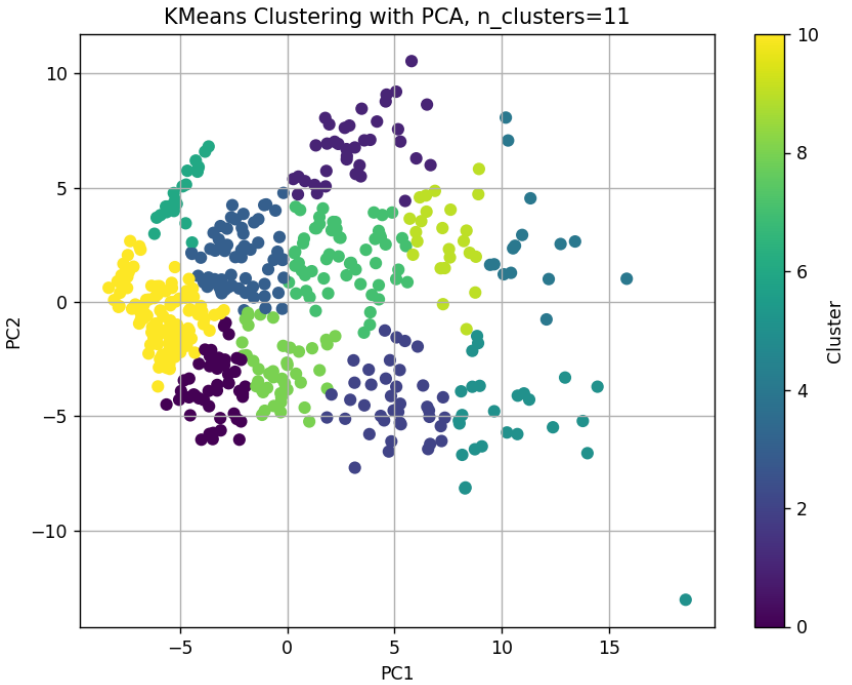
- Cluster cohesion: While the clusters are generally well-separated, there is some overlap, especially when using a smaller number of clusters.

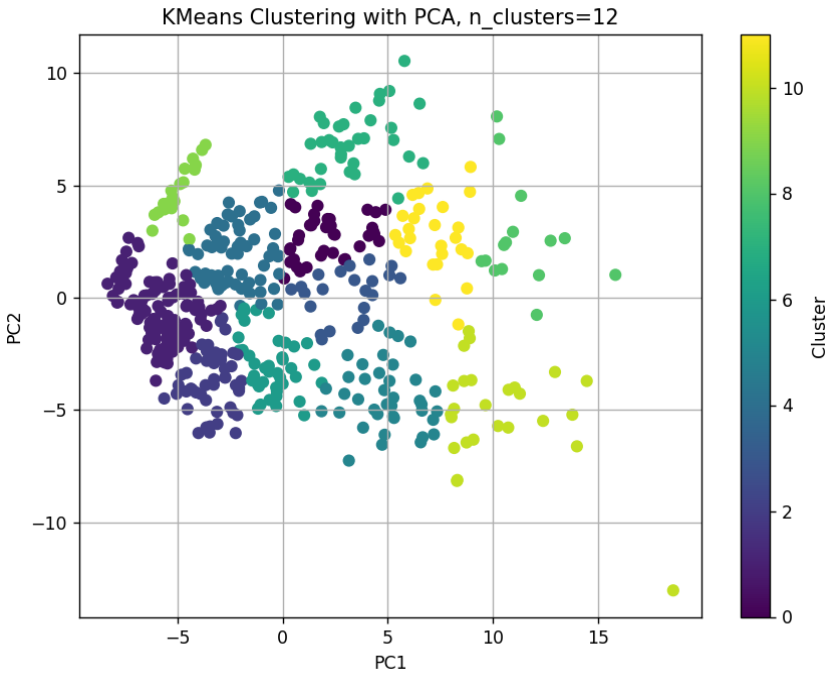







KMeans Clustering with PCA, n_clusters=10
PC1
PC2
Cluster


KMeans Clustering with PCA, n_clusters=11
PC1
PC2
Cluster


KMeans Clustering with PCA, n_clusters=12
PC1
PC2
Cluster

Conclusion

The dimensionality reduction via PCA allows for easier visualization of the clustering results, while K-means clustering groups players based on their statistics.

Number of Clusters: The optimal number of clusters can be chosen based on the business need and interpretability of the results. A smaller number of clusters (e.g., 7-9) is useful for a more general classification, while a higher number (e.g., 10-12) provides finer categorization.

Visual Insights: The 2D visualization of the clusters makes it easier to interpret and compare the player groups based on their performance.

Problem IV:

IV.1: Collecting Player Transfer Values for the 2024-2025 Season

1. Problem Overview

The goal of this task is to collect transfer values for football players from the 2024-2025 season on FootballTransfers. We only want to gather data for players who have played more than 900 minutes during the season. The collected data will be matched with existing player statistics to ensure the accuracy of transfer values.

2. Approach

Step 1: Data Preprocessing

- Text Normalization: A function was created to remove accents and convert player and team names to lowercase. This ensures uniformity in name matching.
- Token-Sorting: To improve name matching accuracy, we sorted tokens within names. This method helps with matching names regardless of their word order.

Step 2: Fuzzy Name Matching

- Fuzzy Matching: The script uses the RapidFuzz library to match player names between two datasets (the transfer data and player statistics). The matching threshold was set to 85 to ensure high accuracy while preventing false positives.

Step 3: Collecting Transfer Data

- API Request: A POST request was sent to FootballTransfers's API to retrieve player transfer values, iterating over multiple pages of results. Data for each player, including their name, team, and estimated transfer value, was collected.

- Concatenation: The player data from all pages was combined into a single dataframe for further processing.

Step 4: Merging with Player Statistics

- Matching Player Names: The player names from the collected transfer data were matched with the names in the CSV file containing player statistics (including minutes played).
- Merging: Using fuzzy matching results, the data from the API was merged with the player statistics dataframe based on matched player names.

Step 5: Filtering by Playing Time

- Minutes Filter: Only players who played more than 900 minutes in the season were kept for further analysis.

Step 6: Handling Missing Transfer Values

- Missing Values: If a transfer value was missing in the merged dataset, an external function (tra_ve_value) was used to look up and complete the missing data. The player names and teams were prepared for URL encoding to query the external source.

Step 7: Exporting Data

- The final filtered and merged dataset, containing player names, teams, minutes played, and transfer values, was saved as a new CSV file for further use or analysis.

3. Code Execution and Output

- Data Collection: Transfer values were collected for players in the 2024-2025 season, with values included for players who met the 900-minute threshold.
- Fuzzy Matching: The fuzzy matching function successfully identified corresponding player names in both datasets, ensuring accurate data merging.
- Final Output: The dataset was filtered and cleaned, with missing transfer values filled. The results were saved into a file called results4.csv.

1	Player	Team	Min	Value
2	Aaron Ramsdale	Southampton	2340	€18.7M
3	Aaron Wan-Bissaka	West Ham	2794	€29.7M
4	Abdoulaye Doucouré	Everton	2425	€5.8M
5	Adam Armstrong	Southampton	1248	€15.1M
6	Adam Smith	Bournemouth	1409	€1.5M
7	Adam Wharton	Crystal Palace	1258	€49.3M
8	Adama Traoré	Fulham	1568	€8.2M
9	Alejandro Garnacho	Manchester Utd	2056	€59.2M
10	Alex Iwobi	Fulham	2721	€31.2M
11	Alexander Isak	Newcastle Utd	2487	€119.4M
12	Alexis Mac Allister	Liverpool	2553	€117M
13	Alisson	Liverpool	2148	€24.5M
14	Alphonse Areola	West Ham	1990	€11.6M

Conclusion

- Data Integration: The process successfully integrated transfer values with player statistics, ensuring that only players who played more than 900 minutes were considered.
- Data Quality: Fuzzy matching, coupled with external value lookups, ensured that the dataset was accurate and complete, even when transfer values were initially missing.
- File Export: The results have been saved as a CSV file (results4.csv) for further analysis or reporting.

IV.2: Football Player Value Prediction

1. Preparation

Step 1: Load and Clean Data

- Load the dataset from a CSV file.
- Filter out players with less than 900 minutes played.
- Replace missing or non-available values (e.g., 'N/a') with 0.

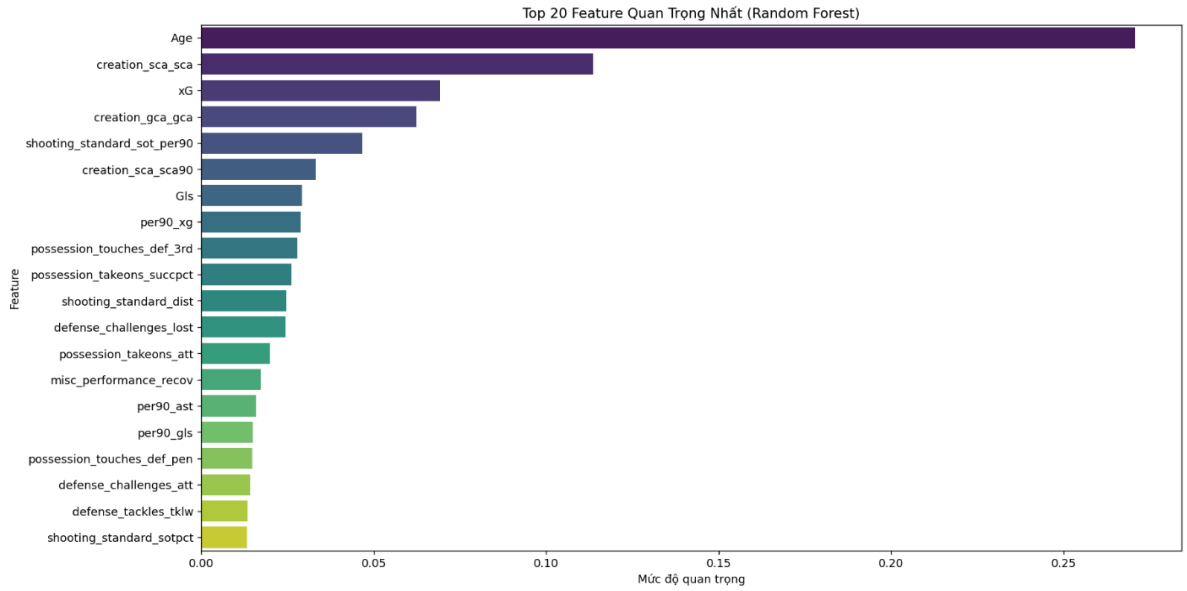
Step 2: Convert Player Value Format

- Transform value strings (e.g., '€20M') into numerical values for analysis.

Step 3: Feature Selection

- Selected features include:
    - Performance metrics (Goals, Assists, xG, xAG)
    - Defensive metrics (Tackles, Aerial Duels Won)
    - Player Age
- Further Feature Selection using Random Forest:

  A Random Forest Regressor was used to compute feature importances and select the top 20 most influential features for predicting player value. This method helps reduce overfitting and focuses the model on the most informative variables.



2. Model Training and Initial Evaluation

Step 1: Train Models

Three different models were initially trained:

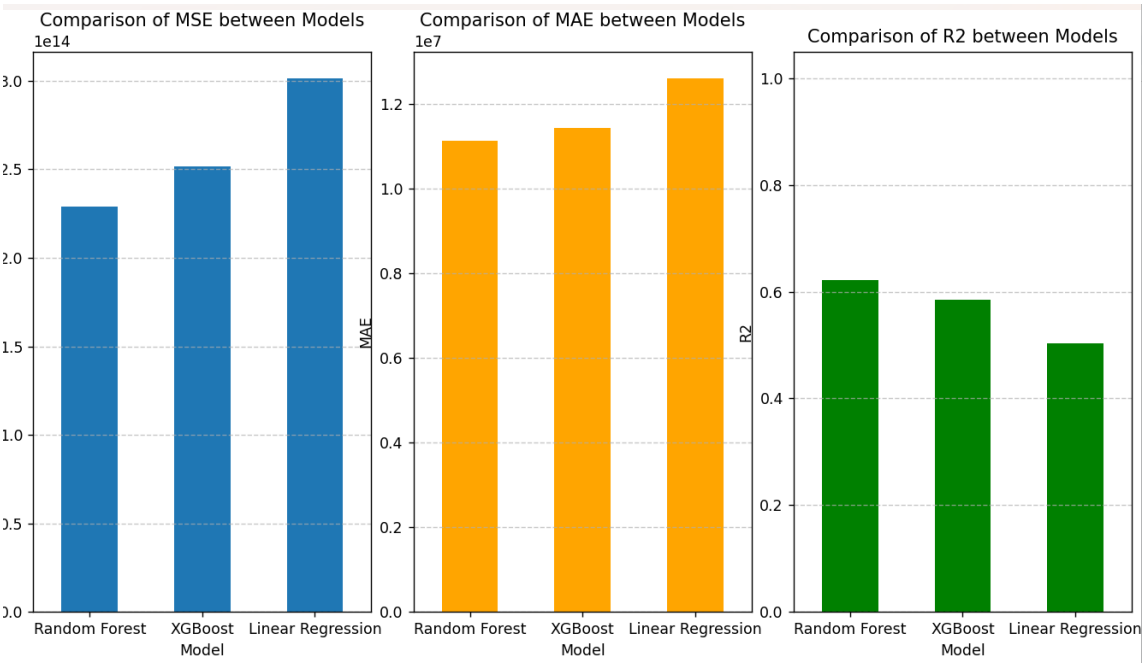
- Linear Regression
- Random Forest Regressor
- XGBoost Regression

Step 2: Evaluate Models

Evaluation was based on:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- R-squared (R^2)

```
--- Final evaluation results of models ---
               Model           MSE           MAE      R2           Params
1      Random Forest  2.290426e+14  1.112075e+07  0.6220  N/A (No tuning)
2            XGBoost  2.514669e+14  1.142777e+07  0.5849  N/A (No tuning)
0  Linear Regression  3.014790e+14  1.260459e+07  0.5024  N/A (No tuning)
```



Step 3: Model Selection

- Random Forest showed the best performance (highest R^2 and lowest errors).
- Therefore, XGBoost was selected for further tuning.

3. Model Tuning

Step 1: Hyperparameter Tuning

- Only XGBoost was tuned to optimize its performance.
- GridSearchCV was used with the following parameters:
    - n_estimators
    - max_depth
    - learning_rate

Step 2: Retraining and Final Evaluation

- The tuned Random Forest model was retrained with the best-found hyperparameters.
- Performance improved slightly compared to the initial model.

```
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=5, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=10, n_estimators=100; total time=   0.2s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=10, n_estimators=100; total time=   0.1s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=10, n_estimators=100; total time=   0.1s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=10, n_estimators=100; total time=   0.1s
[CV] END max_depth=10, max_features=0.5, min_samples_leaf=6, min_samples_split=10, n_estimators=100; total time=   0.1s
  Tuning completed for Random Forest.
  Best parameters: {'max_depth': 5, 'max_features': 0.5, 'min_samples_leaf': 6, 'min_samples_split': 5, 'n_estimators': 100}
  Best R2 score on training set (cross-validation): 0.3929
  Evaluation completed for Random Forest.
  Train results: MSE=245961299846214.62, MAE=10937866.85, R2=0.6442
  Test results : MSE=288347895415238.62, MAE=12679579.99, R2=0.5241
```

Conclusion

After comparing different models and tuning the best one, the tuned Random Forest achieved the highest accuracy in predicting football player values. It is recommended for future deployment. Further improvements could be explored by including additional features like player position, nationality, and injury history