POST AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

PYTHON PROGRAMMING

# PYTHON ASSIGNMENT 1

| | |
|---|---|
| **LECTURER** | : KIM NGOC BACH |
| **STUDENT NAME** | : NGUYEN TIEN THANG |
| **STUDENT ID** | : B23DCCE088 |
| **CLASS** | : D23CQCE04-B |

**Ha Noi – 2025**

## Contents

# PREFACE

This report is the outcome of my efforts for Assignment 1 of the Python Programming course, focused on real-world data collection, statistical analysis, clustering, and predictive modeling using football player statistics from the English Premier League (2024–2025 season). The project offered me an opportunity to integrate multiple programming and data science skills into a single, cohesive analytical workflow.
Throughout this assignment, I worked extensively with web scraping tools such as Selenium and BeautifulSoup to collect rich, structured data from public sports databases. I applied data preprocessing and visualization techniques using pandas, NumPy, and Matplotlib to derive meaningful insights into player performance and team dynamics. By utilizing machine learning methods like KMeans clustering and Principal Component Analysis (PCA), I was able to explore natural groupings among players. Finally, I developed a regression model to estimate player transfer values based on performance statistics.
The completion of this assignment has deepened my understanding of Python's capabilities in data analysis, strengthened my problem-solving skills, and enhanced my appreciation for the complexity and richness of sports analytics. I sincerely hope this report reflects the knowledge, effort, and dedication I have put into this project.

# 1 PART I – DATA COLLECTION

## 1.1 1. Assignment Requirements

- Collect statistical data [*] for all players who have played more than 90 minutes in the 2024-2025 English Premier League season.
- Data source: `https://fbref.com/en/`
- Save the result to a file named 'results.csv', where the result table has the following structure:
  - Each column corresponds to a statistic.
  - Players are sorted alphabetically by their first name.
  - Any statistic that is unavailable or inapplicable should be marked as "N/a".

* The required statistics are:
  - **Nation**
  - **Team**
  - **Position**
  - **Age**
  - **Playing Time**: matches played, starts, minutes
  - **Performance**: goals, assists, yellow cards, red cards
  - **Expected**: expected goals (xG), expedted Assist Goals (xAG)
  - **Progression**: PrgC, PrgP, PrgR
  - **Per 90 minutes**: Gls, Ast, xG, xGA
  - **Goalkeeping**:
    - ∗ Performance: goals against per 90mins (GA90), Save%, CS%
    - ∗ Penalty Kicks: penalty kicks Save%
  - **Shooting**:
    - ∗ Standard: shoots on target percentage (SoT%), Shoot on Target per 90min (SoT/90), goals/shot (G/sh), average shoot distance (Dist)
  - **Passing**:
    - ∗ Total: passes completed (Cmp),Pass completion (Cmp%), progressive passing distance (TotDist)
    - ∗ Short: Pass completion (Cmp%),
    - ∗ Medium: Pass completion (Cmp%),
    - ∗ Long: Pass completion (Cmp%),
    - ∗ Expected: key passes (KP), pass into final third (1/3), pass into penalty area (PPA), CrsPA, PrgP
  - **Goal and Shot Creation**:
    - ∗ SCA: SCA, SCA90
    - ∗ GCA: GCA, GCA90
  - **Defensive Actions**:
    - ∗ Tackles: Tkl, TklW

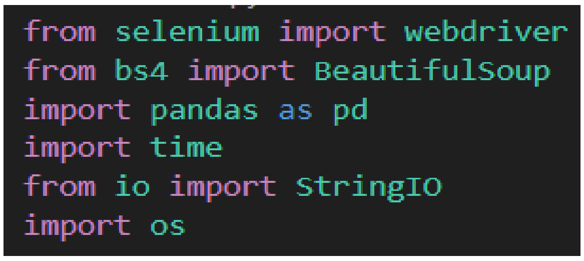
    - Challenges: Att, Lost
    - Blocks: Blocks, Sh, Pass, Int
  - **Possession**:
    - Touches: Touches, Def Pen, Def 3rd, Mid 3rd, Att 3rd, Att Pen
    - Take-Ons: Att, Succ%, Tkld%
    - Carries: Carries, ProDist, ProgC, 1/3, CPA, Mis, Dis
    - Receiving: Rec, PrgR
  - **Miscellaneous Stats**:
    - Performance: Fls, Fld, Off, Crs, Recov
    - Aerial Duels: Won, Lost, Won%

## 1.2 2. Assignment Breakdown

1. **Web Scraping:** Extract player data from the given URLs using Selenium and Beautiful Soup.
2. **Data Extraction:** Extract data into Pandas DataFrames.
3. **Data Merging:** Merge DataFrames on common keys.
4. **Data Filtering:** Filter players based on the "minutes played" criterion.
5. **Data Cleaning:**
   - Handle missing values.
   - Format the data (e.g., convert data types).
6. **Data Structuring:** Organize the data into a Pandas DataFrame.
7. **Sorting:** Sort the DataFrame by the player's first name.
8. **Saving to CSV:** Save the DataFrame to a CSV file.

## 1.3 3. Solution Approach

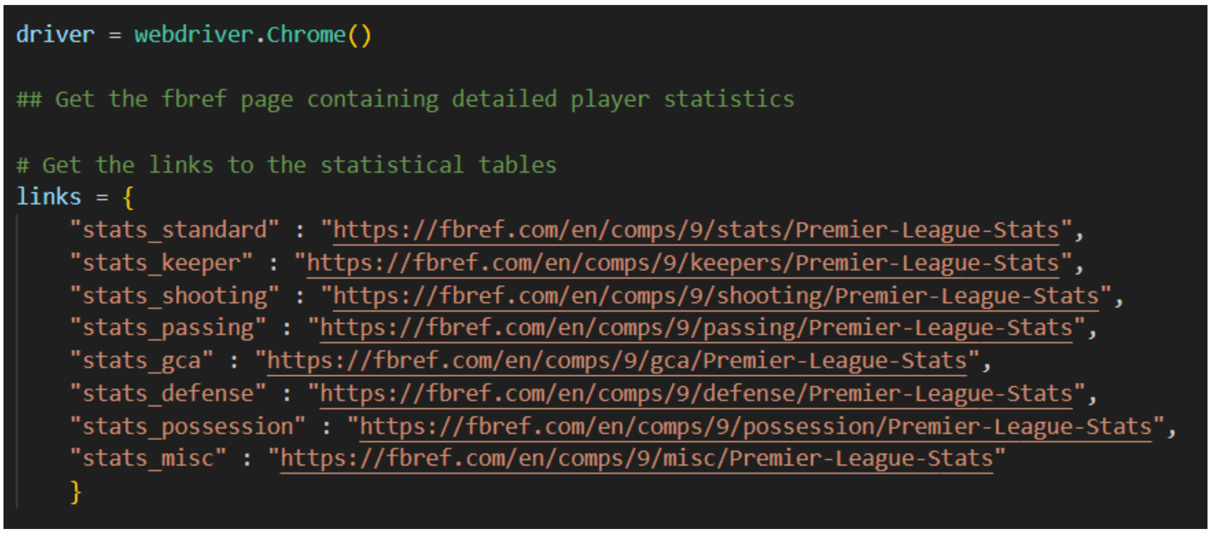
1. **Import necessary libraries.**



```python
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd
import time
from io import StringIO
import os
```

- selenium.webdriver : Automate and control web browser.
- BeautifulSoup : Parse and navigate the HTML content.
- pandas : Handle tabular data: merge, filter, clean, and export.
- time : Pause program execution to allow page loading.
- io.StringIO : Convert string to file-like object for pandas to read.
- os: Interact with the operating system, check for file paths, create folders, and manage directories.

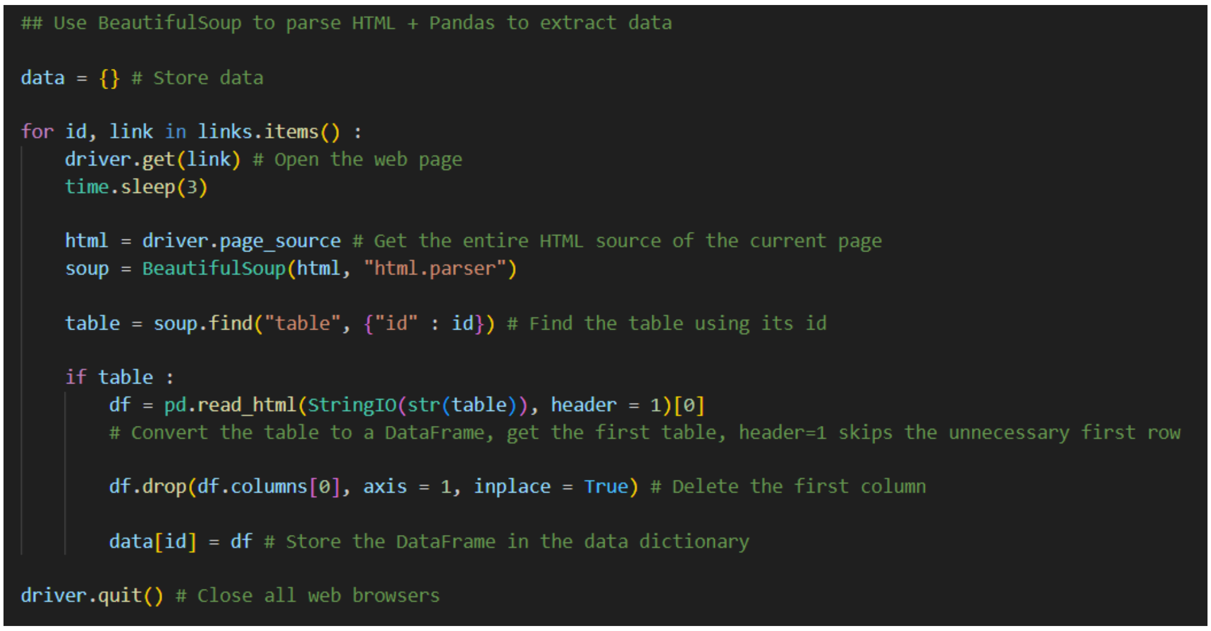
2. **Initialize the webdriver and declare the URLs containing the statistical tables.**



```python
driver = webdriver.Chrome()

## Get the fbref page containing detailed player statistics

# Get the links to the statistical tables
links = {
    "stats_standard" : "https://fbref.com/en/comps/9/stats/Premier-League-Stats",
    "stats_keeper" : "https://fbref.com/en/comps/9/keepers/Premier-League-Stats",
    "stats_shooting" : "https://fbref.com/en/comps/9/shooting/Premier-League-Stats",
    "stats_passing" : "https://fbref.com/en/comps/9/passing/Premier-League-Stats",
    "stats_gca" : "https://fbref.com/en/comps/9/gca/Premier-League-Stats",
    "stats_defense" : "https://fbref.com/en/comps/9/defense/Premier-League-Stats",
    "stats_possession" : "https://fbref.com/en/comps/9/possession/Premier-League-Stats",
    "stats_misc" : "https://fbref.com/en/comps/9/misc/Premier-League-Stats"
    }
```

- Launch the Chrome WebDriver to programmatically control a Chrome browser.
- Declare a dictionary of links to each statistical table on `https://fbref.com/en/`, using the table's HTML id as the key (e.g., `"stats_standard"`, `"stats_shooting"`).
- This organization simplifies parsing and merging later on.

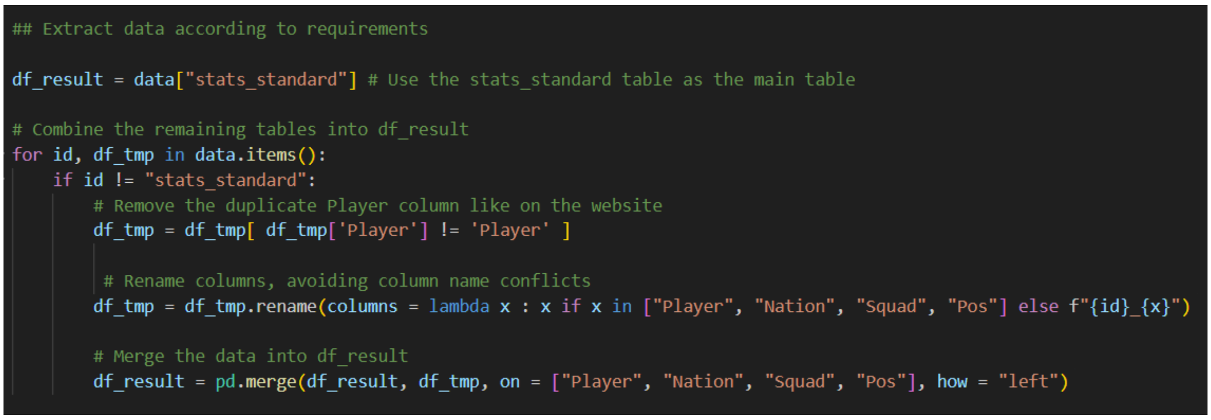
3. **Scrape and Parse HTML Tables using BeautifulSoup and Pandas**



```
## Use BeautifulSoup to parse HTML + Pandas to extract data

data = {} # Store data

for id, link in links.items() :
    driver.get(link) # Open the web page
    time.sleep(3)

    html = driver.page_source # Get the entire HTML source of the current page
    soup = BeautifulSoup(html, "html.parser")

    table = soup.find("table", {"id" : id}) # Find the table using its id

    if table :
        df = pd.read_html(StringIO(str(table)), header = 1)[0]
        # Convert the table to a DataFrame, get the first table, header=1 skips the unnecessary first row

        df.drop(df.columns[0], axis = 1, inplace = True) # Delete the first column

        data[id] = df # Store the DataFrame in the data dictionary

driver.quit() # Close all web browsers
```

- Use Selenium to open each URL and retrieve the full page source after allowing time for full loading.
- Parse the HTML with BeautifulSoup.
- Locate the desired ¡table¿ using its unique ID.
- Extract the table's contents into a Pandas DataFrame.
- Clean the table:
  - Remove the first unnamed index column
  - Store each cleaned DataFrame in a dictionary for merging later.
- Close the browser once all tables have been scraped.

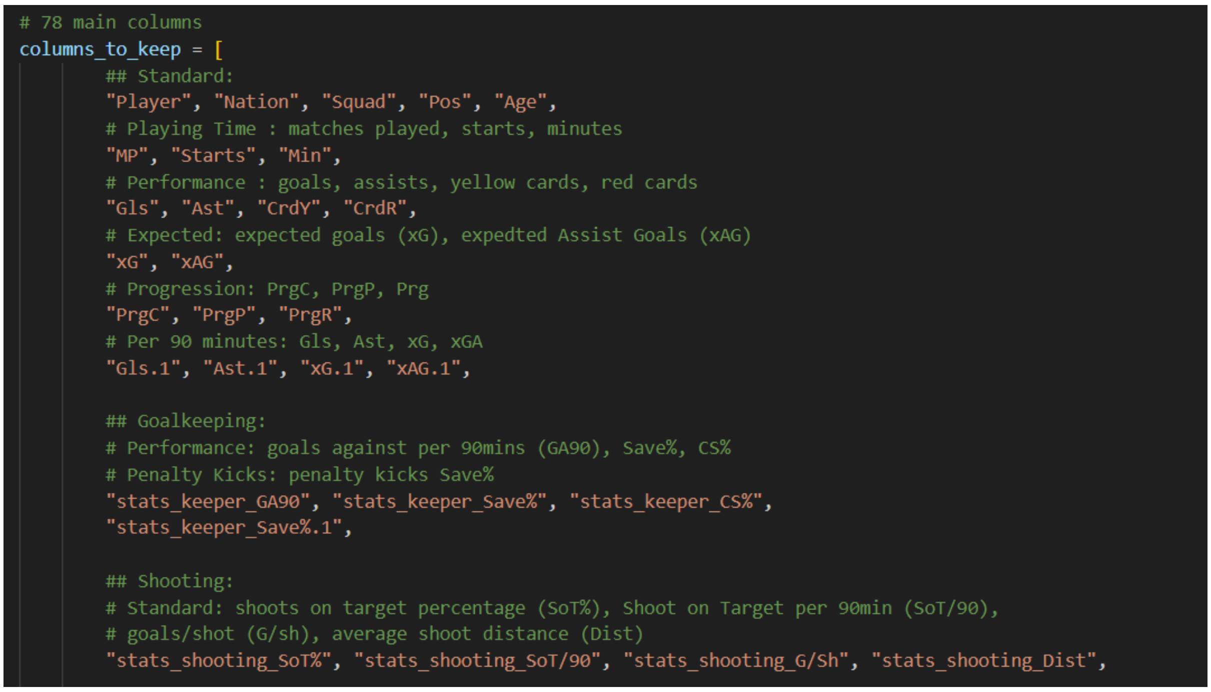
4. **Merge all DataFrames into the main stats_standard table**



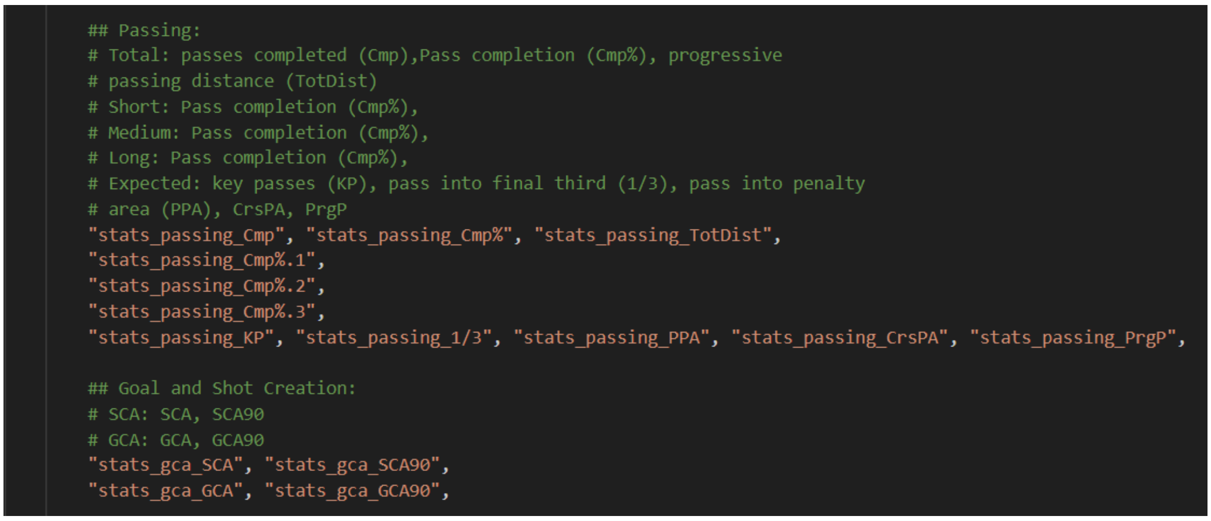
```
## Extract data according to requirements

df_result = data["stats_standard"] # Use the stats_standard table as the main table

# Combine the remaining tables into df_result
for id, df_tmp in data.items():
    if id != "stats_standard":
        # Remove the duplicate Player column like on the website
        df_tmp = df_tmp[ df_tmp['Player'] != 'Player' ]

         # Rename columns, avoiding column name conflicts
        df_tmp = df_tmp.rename(columns = lambda x : x if x in ["Player", "Nation", "Squad", "Pos"] else f"{id}_{x}")

        # Merge the data into df_result
        df_result = pd.merge(df_result, df_tmp, on = ["Player", "Nation", "Squad", "Pos"], how = "left")
```

- Start with the stats_standard table as the base DataFrame. Loop through all other statistical tables:
  - Remove extra header rows (where 'Player' equals 'Player').
  - Rename columns to avoid naming conflicts by prefixing them with the table's id.
  - Merge each cleaned table into the main DataFrame df_result using a left join on Player, Nation, Squad, and Pos columns.

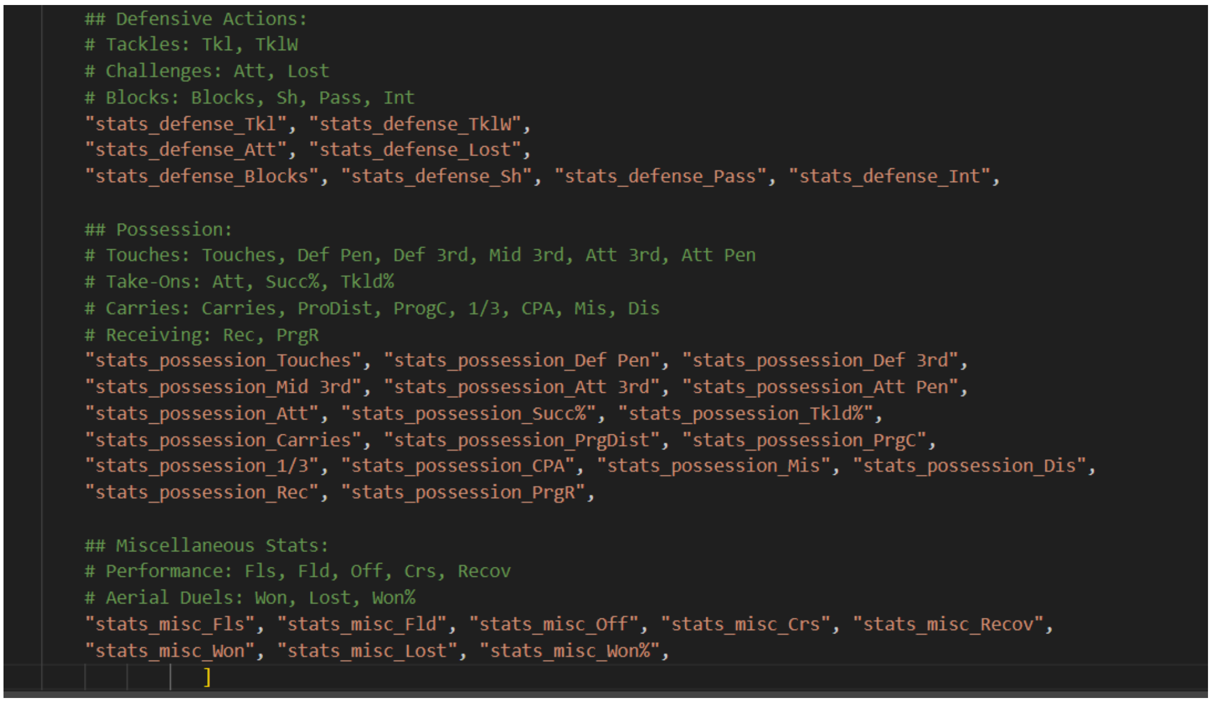
5. **Select 78 specific statistic columns as required by the assignment**



```python
# 78 main columns
columns_to_keep = [
        ## Standard:
        "Player", "Nation", "Squad", "Pos", "Age",
        # Playing Time : matches played, starts, minutes
        "MP", "Starts", "Min",
        # Performance : goals, assists, yellow cards, red cards
        "Gls", "Ast", "CrdY", "CrdR",
        # Expected: expected goals (xG), expedted Assist Goals (xAG)
        "xG", "xAG",
        # Progression: PrgC, PrgP, Prg
        "PrgC", "PrgP", "PrgR",
        # Per 90 minutes: Gls, Ast, xG, xGA
        "Gls.1", "Ast.1", "xG.1", "xAG.1",

        ## Goalkeeping:
        # Performance: goals against per 90mins (GA90), Save%, CS%
        # Penalty Kicks: penalty kicks Save%
        "stats_keeper_GA90", "stats_keeper_Save%", "stats_keeper_CS%",
        "stats_keeper_Save%.1",

        ## Shooting:
        # Standard: shoots on target percentage (SoT%), Shoot on Target per 90min (SoT/90),
        # goals/shot (G/sh), average shoot distance (Dist)
        "stats_shooting_SoT%", "stats_shooting_SoT/90", "stats_shooting_G/Sh", "stats_shooting_Dist",
```



```python
        ## Passing:
        # Total: passes completed (Cmp),Pass completion (Cmp%), progressive
        # passing distance (TotDist)
        # Short: Pass completion (Cmp%),
        # Medium: Pass completion (Cmp%),
        # Long: Pass completion (Cmp%),
        # Expected: key passes (KP), pass into final third (1/3), pass into penalty
        # area (PPA), CrsPA, PrgP
        "stats_passing_Cmp", "stats_passing_Cmp%", "stats_passing_TotDist",
        "stats_passing_Cmp%.1",
        "stats_passing_Cmp%.2",
        "stats_passing_Cmp%.3",
        "stats_passing_KP", "stats_passing_1/3", "stats_passing_PPA", "stats_passing_CrsPA", "stats_passing_PrgP",

        ## Goal and Shot Creation:
        # SCA: SCA, SCA90
        # GCA: GCA, GCA90
        "stats_gca_SCA", "stats_gca_SCA90",
        "stats_gca_GCA", "stats_gca_GCA90",
```

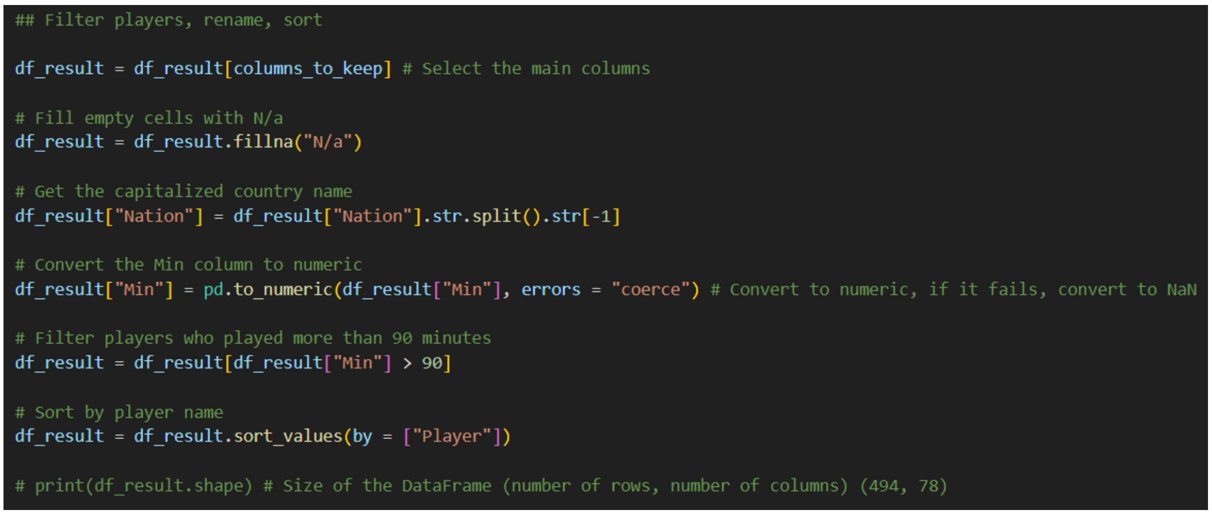


```
    ## Defensive Actions:
    # Tackles: Tkl, TklW
    # Challenges: Att, Lost
    # Blocks: Blocks, Sh, Pass, Int
    "stats_defense_Tkl", "stats_defense_TklW",
    "stats_defense_Att", "stats_defense_Lost",
    "stats_defense_Blocks", "stats_defense_Sh", "stats_defense_Pass", "stats_defense_Int",

    ## Possession:
    # Touches: Touches, Def Pen, Def 3rd, Mid 3rd, Att 3rd, Att Pen
    # Take-Ons: Att, Succ%, Tkld%
    # Carries: Carries, ProDist, ProgC, 1/3, CPA, Mis, Dis
    # Receiving: Rec, PrgR
    "stats_possession_Touches", "stats_possession_Def Pen", "stats_possession_Def 3rd",
    "stats_possession_Mid 3rd", "stats_possession_Att 3rd", "stats_possession_Att Pen",
    "stats_possession_Att", "stats_possession_Succ%", "stats_possession_Tkld%",
    "stats_possession_Carries", "stats_possession_PrgDist", "stats_possession_PrgC",
    "stats_possession_1/3", "stats_possession_CPA", "stats_possession_Mis", "stats_possession_Dis",
    "stats_possession_Rec", "stats_possession_PrgR",

    ## Miscellaneous Stats:
    # Performance: Fls, Fld, Off, Crs, Recov
    # Aerial Duels: Won, Lost, Won%
    "stats_misc_Fls", "stats_misc_Fld", "stats_misc_Off", "stats_misc_Crs", "stats_misc_Recov",
    "stats_misc_Won", "stats_misc_Lost", "stats_misc_Won%",
        ]
```

- Create a list named `columns_to_keep` containing 78 specific columns from different statistical tables. These include:
  - player information
  - performance stats
  - goalkeeping
  - shooting
  - passing
  - defensive actions
  - possession
  - miscellaneous stats

  (Selected based on the assignment's requirements)
- This ensures only relevant data will be kept for analysis and CSV export later.

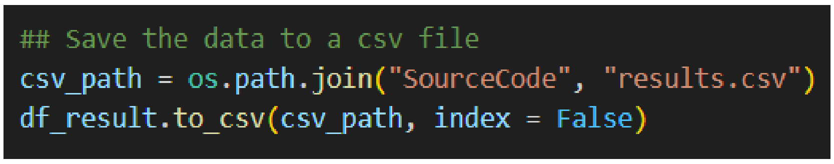
6. **Clean and process the data**



```python
## Filter players, rename, sort

df_result = df_result[columns_to_keep] # Select the main columns

# Fill empty cells with N/a
df_result = df_result.fillna("N/a")

# Get the capitalized country name
df_result["Nation"] = df_result["Nation"].str.split().str[-1]

# Convert the Min column to numeric
df_result["Min"] = pd.to_numeric(df_result["Min"], errors = "coerce") # Convert to numeric, if it fails, convert to NaN

# Filter players who played more than 90 minutes
df_result = df_result[df_result["Min"] > 90]

# Sort by player name
df_result = df_result.sort_values(by = ["Player"])

# print(df_result.shape) # Size of the DataFrame (number of rows, number of columns) (494, 78)
```

- This step selects the required 78 columns.
- Fill all missing or blank values with ”N/a” for consistency.
- Extract the capitalized nationality code from the ’Nation’ column.
- Convert the ’Min’ (minutes played) field to numeric and drop rows where value $\leq 90$.
- Sort the DataFrame alphabetically by ’Player’ for easier readability and comparison.

7. **Save the data**

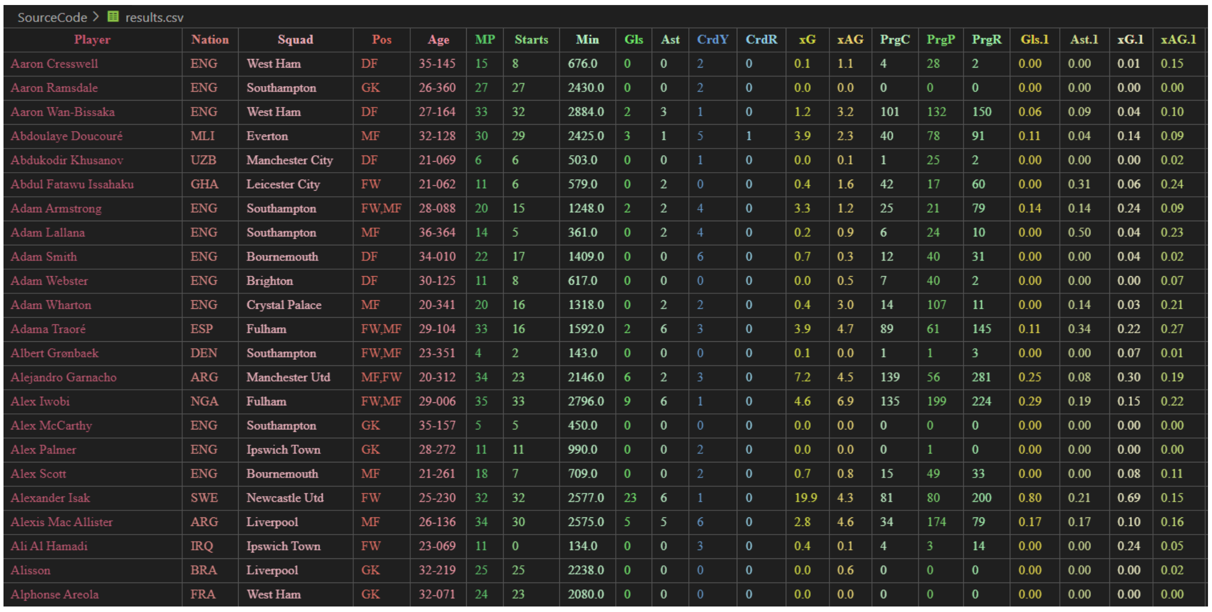


```python
## Save the data to a csv file
csv_path = os.path.join("SourceCode", "results.csv")
df_result.to_csv(csv_path, index = False)
```

- These lines saves the cleaned and processed DataFrame to a CSV file named results.csv without including the row index.

## 1.4 4. Results

- The primary result of Part I is the creation of the 'results.csv' file.
- This CSV file contains player statistics for the 2024-2025 English Premier League season, including data on player information, playing time, performance, and more.
- The data is organized in a table format, with each row representing a player and each column representing a statistic.
- Players are sorted alphabetically, and missing data is represented by 'N/a'.
- Example :



SourceCode > results.csv

| Player | Nation | Squad | Pos | Age | MP | Starts | Min | Gls | Ast | CrdY | CrdR | xG | xAG | PrgC | PrgP | PrgR | Gls.1 | Ast.1 | xG.1 | xAG.1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aaron Cresswell | ENG | West Ham | DF | 35-145 | 15 | 8 | 676.0 | 0 | 0 | 2 | 0 | 0.1 | 1.1 | 4 | 28 | 2 | 0.00 | 0.00 | 0.01 | 0.15 |
| Aaron Ramsdale | ENG | Southampton | GK | 26-360 | 27 | 27 | 2430.0 | 0 | 0 | 2 | 0 | 0.0 | 0.0 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Aaron Wan-Bissaka | ENG | West Ham | DF | 27-164 | 33 | 32 | 2884.0 | 2 | 3 | 1 | 0 | 1.2 | 3.2 | 101 | 132 | 150 | 0.06 | 0.09 | 0.04 | 0.10 |
| Abdoulaye Doucouré | MLI | Everton | MF | 32-128 | 30 | 29 | 2425.0 | 3 | 1 | 5 | 1 | 3.9 | 2.3 | 40 | 78 | 91 | 0.11 | 0.04 | 0.14 | 0.09 |
| Abdukodir Khusanov | UZB | Manchester City | DF | 21-069 | 6 | 6 | 503.0 | 0 | 0 | 1 | 0 | 0.0 | 0.1 | 1 | 25 | 2 | 0.00 | 0.00 | 0.00 | 0.02 |
| Abdul Fatawu Issahaku | GHA | Leicester City | FW | 21-062 | 11 | 6 | 579.0 | 0 | 2 | 0 | 0 | 0.4 | 1.6 | 42 | 17 | 60 | 0.00 | 0.31 | 0.06 | 0.24 |
| Adam Armstrong | ENG | Southampton | FW,MF | 28-088 | 20 | 15 | 1248.0 | 2 | 2 | 4 | 0 | 3.3 | 1.2 | 25 | 21 | 79 | 0.14 | 0.14 | 0.24 | 0.09 |
| Adam Lallana | ENG | Southampton | MF | 36-364 | 14 | 5 | 361.0 | 0 | 2 | 4 | 0 | 0.2 | 0.9 | 6 | 24 | 10 | 0.00 | 0.50 | 0.04 | 0.23 |
| Adam Smith | ENG | Bournemouth | DF | 34-010 | 22 | 17 | 1409.0 | 0 | 0 | 6 | 0 | 0.7 | 0.3 | 12 | 40 | 31 | 0.00 | 0.00 | 0.04 | 0.02 |
| Adam Webster | ENG | Brighton | DF | 30-125 | 11 | 8 | 617.0 | 0 | 0 | 0 | 0 | 0.0 | 0.5 | 7 | 40 | 2 | 0.00 | 0.00 | 0.00 | 0.07 |
| Adam Wharton | ENG | Crystal Palace | MF | 20-341 | 20 | 16 | 1318.0 | 0 | 2 | 2 | 0 | 0.4 | 3.0 | 14 | 107 | 11 | 0.00 | 0.14 | 0.03 | 0.21 |
| Adama Traoré | ESP | Fulham | FW,MF | 29-104 | 33 | 16 | 1592.0 | 2 | 6 | 3 | 0 | 3.9 | 4.7 | 89 | 61 | 145 | 0.11 | 0.34 | 0.22 | 0.27 |
| Albert Grønbaek | DEN | Southampton | FW,MF | 23-351 | 4 | 2 | 143.0 | 0 | 0 | 0 | 0 | 0.1 | 0.0 | 1 | 1 | 3 | 0.00 | 0.00 | 0.07 | 0.01 |
| Alejandro Garnacho | ARG | Manchester Utd | MF,FW | 20-312 | 34 | 23 | 2146.0 | 6 | 2 | 3 | 0 | 7.2 | 4.5 | 139 | 56 | 281 | 0.25 | 0.08 | 0.30 | 0.19 |
| Alex Iwobi | NGA | Fulham | FW,MF | 29-006 | 35 | 33 | 2796.0 | 9 | 6 | 1 | 0 | 4.6 | 6.9 | 135 | 199 | 224 | 0.29 | 0.19 | 0.15 | 0.22 |
| Alex McCarthy | ENG | Southampton | GK | 35-157 | 5 | 5 | 450.0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Alex Palmer | ENG | Ipswich Town | GK | 28-272 | 11 | 11 | 990.0 | 0 | 0 | 2 | 0 | 0.0 | 0.0 | 0 | 1 | 0 | 0.00 | 0.00 | 0.00 | 0.00 |
| Alex Scott | ENG | Bournemouth | MF | 21-261 | 18 | 7 | 709.0 | 0 | 0 | 2 | 0 | 0.7 | 0.8 | 15 | 49 | 33 | 0.00 | 0.00 | 0.08 | 0.11 |
| Alexander Isak | SWE | Newcastle Utd | FW | 25-230 | 32 | 32 | 2577.0 | 23 | 6 | 1 | 0 | 19.9 | 4.3 | 81 | 80 | 200 | 0.80 | 0.21 | 0.69 | 0.15 |
| Alexis Mac Allister | ARG | Liverpool | MF | 26-136 | 34 | 30 | 2575.0 | 5 | 5 | 6 | 0 | 2.8 | 4.6 | 34 | 174 | 79 | 0.17 | 0.17 | 0.10 | 0.16 |
| Ali Al Hamadi | IRQ | Ipswich Town | FW | 23-069 | 11 | 0 | 134.0 | 0 | 0 | 3 | 0 | 0.4 | 0.1 | 4 | 3 | 14 | 0.00 | 0.00 | 0.24 | 0.05 |
| Alisson | BRA | Liverpool | GK | 32-219 | 25 | 25 | 2238.0 | 0 | 0 | 0 | 0 | 0.0 | 0.6 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 | 0.02 |
| Alphonse Areola | FRA | West Ham | GK | 32-071 | 24 | 23 | 2080.0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | 0 | 0.00 | 0.00 | 0.00 | 0.00 |

# 2 PART II – DATA ANALYSIS

## 2.1 1. Assignment Requirements

- Identify the top 3 players with the highest and lowest scores for each statistic. Save result to a file name 'top_3.txt'
- Find the median for each statistic. Calculate the mean and standard deviation for each statistic across all players and for each team. Save the results to a file named 'results2.csv' with the following format:

| | | Median of Att tribute 1 | Mean of Att tribute 1 | Std of Att tribute 1 | ... | ... |
|---|---|---|---|---|---|---|
| 0 | all | | | | | |
| 1 | Team 1 | | | | | |
| ... | | | | | | |
| n | Team n | | | | | |

- Plot a histogram showing the distribution of each statistic for all players in the league and each team.
- Identify the team with the highest scores for each statistic. Based on your analysis, which team do you think is performing the best in the 2024-2025 Premier League season?

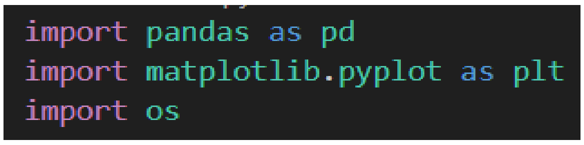
## 2.2 2. Assignment Breakdown

1. **Load Data:** Load the "results.csv" file into a Pandas DataFrame.
2. **Data Preprocessing:** Clean and transform the data as needed (e.g., handle data types).
3. **Top/Bottom Players:** Find and save the top 3 highest and lowest players for each statistic.
4. **Descriptive Statistics:** Calculate and save the median, mean, and standard deviation for each statistic.
5. **Histograms:** Generate histograms to visualize the distribution of statistics.
6. **Team Performance:** Identify the best-performing team based on the statistics.

## 2.3 3. Solution Approach

1. **Data Preparation.**

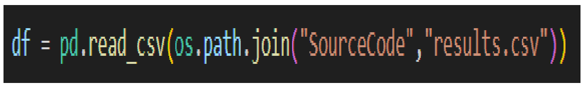
   - **Import necessary libraries.**



```
import pandas as pd
import matplotlib.pyplot as plt
import os
```

     – pandas is used for data manipulation and analysis (like reading CSV files, transforming data, etc.).
     – matplotlib.pyplot is used to create visualizations like histograms.
     – os: Interact with the operating system, check for file paths, create folders, and manage directories.

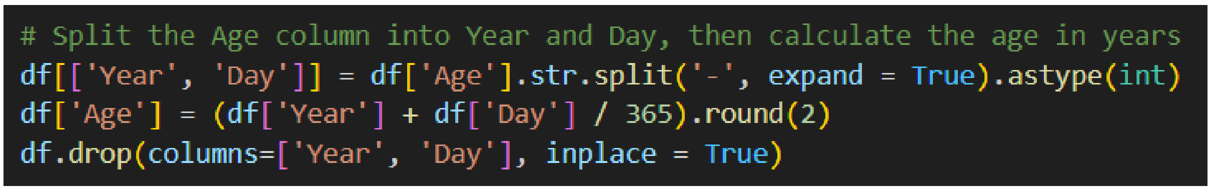
   - **Read and Preprocess Data.**



```
df = pd.read_csv(os.path.join("SourceCode","results.csv"))
```

     – Reads a CSV file named results.csv into a DataFrame called df.

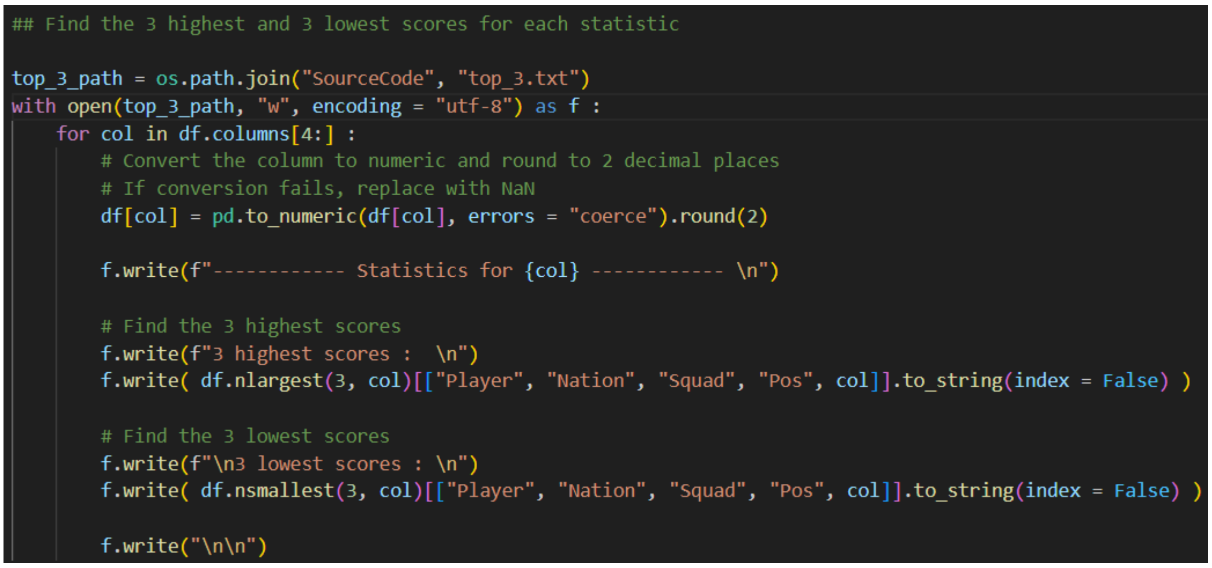
   - **Preprocess Age Data.**



```
# Split the Age column into Year and Day, then calculate the age in years
df[['Year', 'Day']] = df['Age'].str.split('-', expand = True).astype(int)
df['Age'] = (df['Year'] + df['Day'] / 365).round(2)
df.drop(columns=['Year', 'Day'], inplace = True)
```

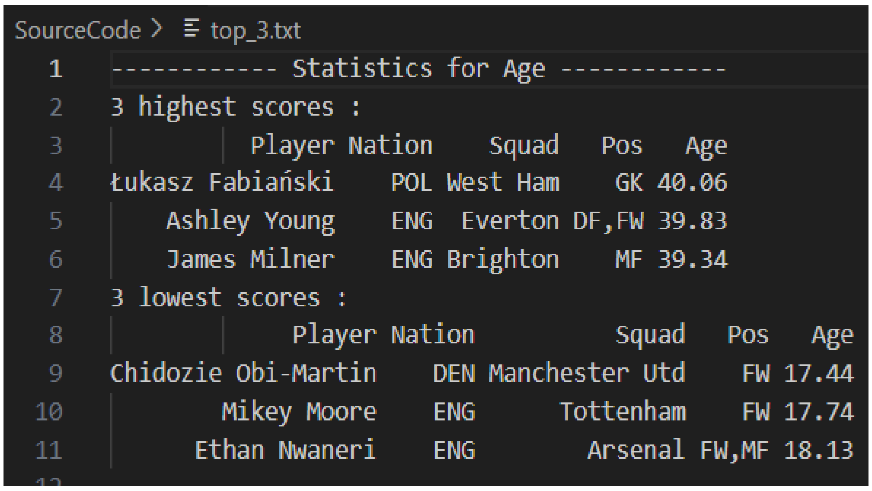
     – The Age column is initially in the format XX-YY (e.g. 24-192 meaning 24 years and 192 days).
     – This code splits it into two columns: Year and Day.
     – Then it calculates the exact age in years with decimal and replaces the original Age column.

2. **Top 3 highest and 3 lowest scores for each statistic.**



```python
## Find the 3 highest and 3 lowest scores for each statistic

top_3_path = os.path.join("SourceCode", "top_3.txt")
with open(top_3_path, "w", encoding = "utf-8") as f :
    for col in df.columns[4:] :
        # Convert the column to numeric and round to 2 decimal places
        # If conversion fails, replace with NaN
        df[col] = pd.to_numeric(df[col], errors = "coerce").round(2)

        f.write(f"----------- Statistics for {col} ----------- \n")

        # Find the 3 highest scores
        f.write(f"3 highest scores :  \n")
        f.write( df.nlargest(3, col)[["Player", "Nation", "Squad", "Pos", col]].to_string(index = False) )

        # Find the 3 lowest scores
        f.write(f"\n3 lowest scores : \n")
        f.write( df.nsmallest(3, col)[["Player", "Nation", "Squad", "Pos", col]].to_string(index = False) )

        f.write("\n\n")
```

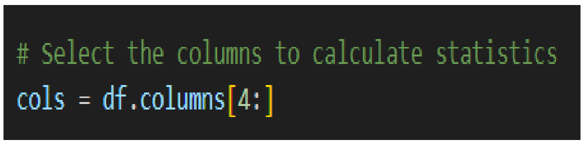
- **Iterate Through Statistical Columns to Determine the Top Team**
  - Iterate through all statistical columns, starting from the 5th column (after `Player`, `Nation`, `Squad`, `Pos`)
  - Convert each column to numeric values using:
    - ∗ `pd.to_numeric()` function
    - ∗ Replace invalid entries with `NaN`
    - ∗ Round all values to 2 decimal places for consistency
  - For each column:
    - ∗ Use `nlargest(3, col)` to identify the top 3 players
    - ∗ Use `nsmallest(3, col)` to identify the bottom 3 players
- **Export to Text File.**
  - Write the results to top_3.txt with clear formatting, including player name, nationality, team, position, and the corresponding value.
- **Example for Results.**



```
------------ Statistics for Age ------------
3 highest scores :
          Player Nation    Squad   Pos   Age
Łukasz Fabiański    POL West Ham    GK 40.06
    Ashley Young    ENG  Everton DF,FW 39.83
    James Milner    ENG Brighton    MF 39.34
3 lowest scores :
             Player Nation          Squad   Pos   Age
Chidozie Obi-Martin    DEN Manchester Utd    FW 17.44
        Mikey Moore    ENG      Tottenham    FW 17.74
      Ethan Nwaneri    ENG        Arsenal FW,MF 18.13
```

3. **The median for each statistic. The mean and standard deviation for each statistic across all players and for each team.**

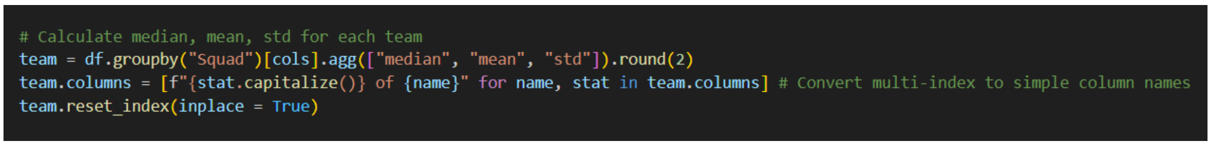
   - **Select Statistical Columns.**



```python
# Select the columns to calculate statistics
cols = df.columns[4:]
```

     – We exclude the first 4 columns (usually identifiers like Player, Nation, Squad, Pos) and only keep the statistical performance metrics.

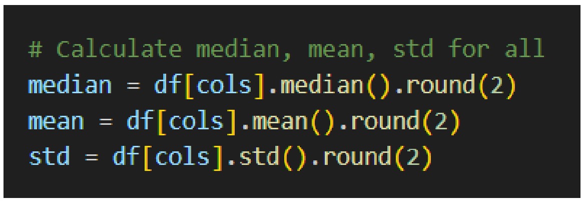
   - **Calculate Median, Mean, and Standard Deviation for Each Team.**



```python
# Calculate median, mean, std for each team
team = df.groupby("Squad")[cols].agg(["median", "mean", "std"]).round(2)
team.columns = [f"{stat.capitalize()} of {name}" for name, stat in team.columns] # Convert multi-index to simple column names
team.reset_index(inplace = True)
```

     – We group the data by Squad (team name), then compute median, mean, and standard deviation for each statistic. The multi-level column names created by .agg() are flattened into readable names (like ”Mean of Goals”), and we reset the index for a clean tabular structure.

   - **Calculate Overall Median, Mean, and Standard Deviation for All Players.**

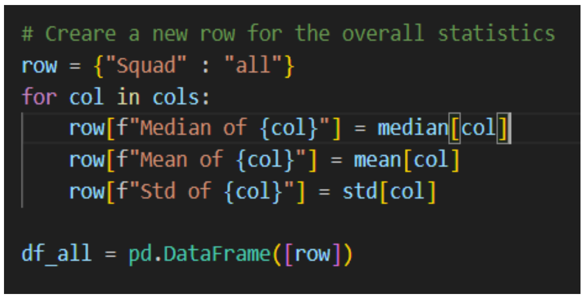


```python
# Calculate median, mean, std for all
median = df[cols].median().round(2)
mean = df[cols].mean().round(2)
std = df[cols].std().round(2)
```

     – Here, the code computes global statistics across all players (regardless of team):
       * median: Middle value of each column
       * mean: Average of each column
       * std: Standard deviation of each column

     – All results are rounded to two decimal places.

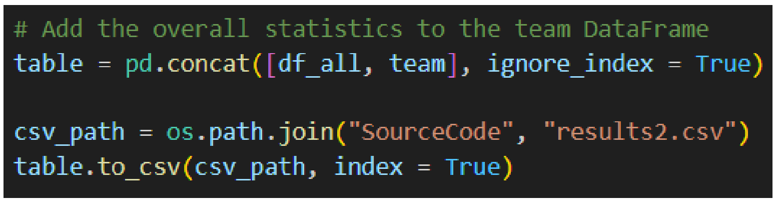
- **Create a Row for Overall Dataset Summary.**



```python
# Creare a new row for the overall statistics
row = {"Squad" : "all"}
for col in cols:
    row[f"Median of {col}"] = median[col]
    row[f"Mean of {col}"] = mean[col]
    row[f"Std of {col}"] = std[col]

df_all = pd.DataFrame([row])
```

  - A dictionary named row is initialized to represent a summary of the entire dataset, labeled with "Squad": "all".
  - For each column (statistic), the corresponding median, mean, and standard deviation are added to the dictionary.
  - This dictionary is then converted into a one-row DataFrame (df_all) to be appended later.

- **Merge and Export the Final Table.**

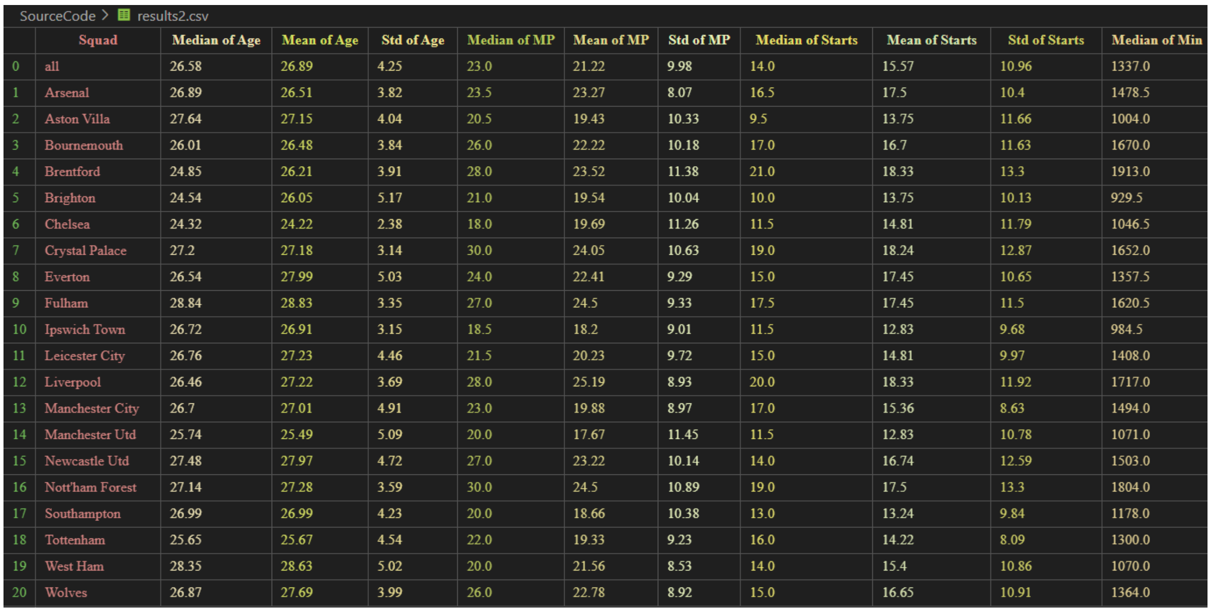


```python
# Add the overall statistics to the team DataFrame
table = pd.concat([df_all, team], ignore_index = True)

csv_path = os.path.join("SourceCode", "results2.csv")
table.to_csv(csv_path, index = True)
```

  - `pd.concat([df_all, team], ignore_index=True)` combines:
    - The overall statistics row (`df_all`)
    - The team-specific statistics table (`team`)

    into one unified table.
  - `ignore_index=True` ensures the row index is continuous and properly ordered.
  - `to_csv()` writes the final result to `results2.csv` for:
    - Reporting
    - Visualization
    - Further analysis

- **Example for Results.**



SourceCode > results2.csv

| | Squad | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | all | 26.58 | 26.89 | 4.25 | 23.0 | 21.22 | 9.98 | 14.0 | 15.57 | 10.96 | 1337.0 |
| 1 | Arsenal | 26.89 | 26.51 | 3.82 | 23.5 | 23.27 | 8.07 | 16.5 | 17.5 | 10.4 | 1478.5 |
| 2 | Aston Villa | 27.64 | 27.15 | 4.04 | 20.5 | 19.43 | 10.33 | 9.5 | 13.75 | 11.66 | 1004.0 |
| 3 | Bournemouth | 26.01 | 26.48 | 3.84 | 26.0 | 22.22 | 10.18 | 17.0 | 16.7 | 11.63 | 1670.0 |
| 4 | Brentford | 24.85 | 26.21 | 3.91 | 28.0 | 23.52 | 11.38 | 21.0 | 18.33 | 13.3 | 1913.0 |
| 5 | Brighton | 24.54 | 26.05 | 5.17 | 21.0 | 19.54 | 10.04 | 10.0 | 13.75 | 10.13 | 929.5 |
| 6 | Chelsea | 24.32 | 24.22 | 2.38 | 18.0 | 19.69 | 11.26 | 11.5 | 14.81 | 11.79 | 1046.5 |
| 7 | Crystal Palace | 27.2 | 27.18 | 3.14 | 30.0 | 24.05 | 10.63 | 19.0 | 18.24 | 12.87 | 1652.0 |
| 8 | Everton | 26.54 | 27.99 | 5.03 | 24.0 | 22.41 | 9.29 | 15.0 | 17.45 | 10.65 | 1357.5 |
| 9 | Fulham | 28.84 | 28.83 | 3.35 | 27.0 | 24.5 | 9.33 | 17.5 | 17.45 | 11.5 | 1620.5 |
| 10 | Ipswich Town | 26.72 | 26.91 | 3.15 | 18.5 | 18.2 | 9.01 | 11.5 | 12.83 | 9.68 | 984.5 |
| 11 | Leicester City | 26.76 | 27.23 | 4.46 | 21.5 | 20.23 | 9.72 | 15.0 | 14.81 | 9.97 | 1408.0 |
| 12 | Liverpool | 26.46 | 27.22 | 3.69 | 28.0 | 25.19 | 8.93 | 20.0 | 18.33 | 11.92 | 1717.0 |
| 13 | Manchester City | 26.7 | 27.01 | 4.91 | 23.0 | 19.88 | 8.97 | 17.0 | 15.36 | 8.63 | 1494.0 |
| 14 | Manchester Utd | 25.74 | 25.49 | 5.09 | 20.0 | 17.67 | 11.45 | 11.5 | 12.83 | 10.78 | 1071.0 |
| 15 | Newcastle Utd | 27.48 | 27.97 | 4.72 | 27.0 | 23.22 | 10.14 | 14.0 | 16.74 | 12.59 | 1503.0 |
| 16 | Nott'ham Forest | 27.14 | 27.28 | 3.59 | 30.0 | 24.5 | 10.89 | 19.0 | 17.5 | 13.3 | 1804.0 |
| 17 | Southampton | 26.99 | 26.99 | 4.23 | 20.0 | 18.66 | 10.38 | 13.0 | 13.24 | 9.84 | 1178.0 |
| 18 | Tottenham | 25.65 | 25.67 | 4.54 | 22.0 | 19.33 | 9.23 | 16.0 | 14.22 | 8.09 | 1300.0 |
| 19 | West Ham | 28.35 | 28.63 | 5.02 | 20.0 | 21.56 | 8.53 | 14.0 | 15.4 | 10.86 | 1070.0 |
| 20 | Wolves | 26.87 | 27.69 | 3.99 | 26.0 | 22.78 | 8.92 | 15.0 | 16.65 | 10.91 | 1364.0 |

4. **Plot a histogram showing the distribution of each statistic for all players in the league and each team ( 3 statistics for attacking and 3 statistics for defending).**

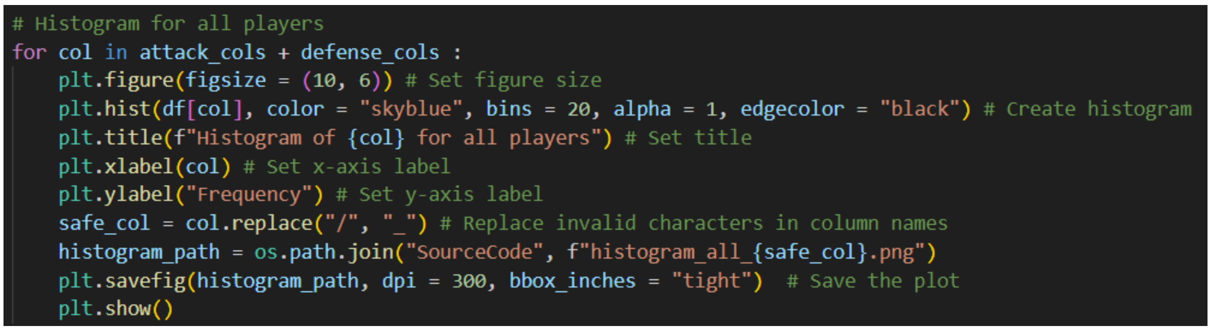
   - **Select Statistics to Visualize.**

```
# 3 statistics for attacking : goals, assists, goals/shot : stats_shooting_sG/sh
attack_cols = ["Gls", "Ast", "stats_shooting_G/Sh"]

# 3 statistics for defending : Tackles: stats_defense_Tkl, stats_defense_TklW. Blocks : stats_defense_Blocks
defense_cols = ["stats_defense_Tkl", "stats_defense_TklW", "stats_defense_Blocks"]
```

     – We manually define two sets of statistics:
       - **Attacking stats:**
         - `Gls` (Goals)
         - `Ast` (Assists)
         - `stats_shooting_G/Sh` (Goals per Shot)
       - **Defending stats:**
         - `stats_defense_Tkl` (Tackles)
         - `stats_defense_TklW` (Successful Tackles)
         - `stats_defense_Blocks` (Blocks)

       These represent key performance indicators (KPIs) for evaluating:
       - Offensive capabilities
       - Defensive contributions

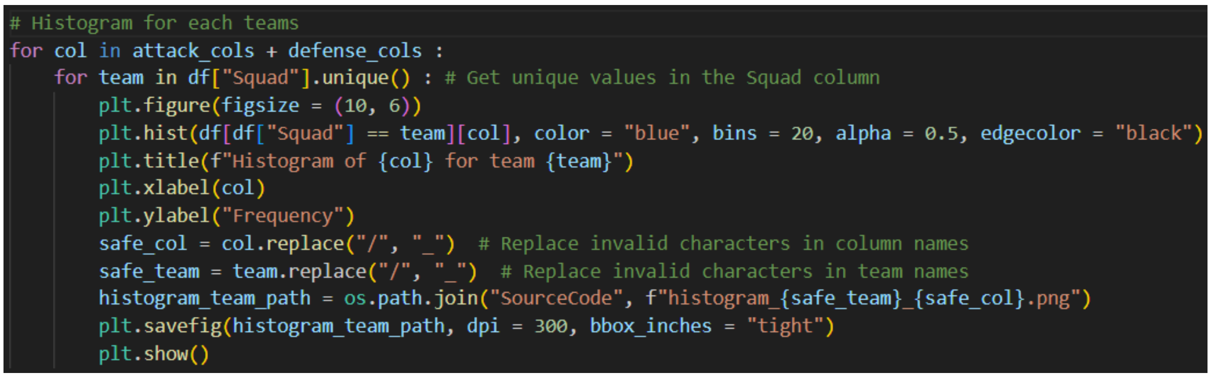
- **Plot Histograms for All Players.**



```python
# Histogram for all players
for col in attack_cols + defense_cols :
    plt.figure(figsize = (10, 6)) # Set figure size
    plt.hist(df[col], color = "skyblue", bins = 20, alpha = 1, edgecolor = "black") # Create histogram
    plt.title(f"Histogram of {col} for all players") # Set title
    plt.xlabel(col) # Set x-axis label
    plt.ylabel("Frequency") # Set y-axis label
    safe_col = col.replace("/", "_") # Replace invalid characters in column names
    histogram_path = os.path.join("SourceCode", f"histogram_all_{safe_col}.png")
    plt.savefig(histogram_path, dpi = 300, bbox_inches = "tight")  # Save the plot
    plt.show()
```

  – We loop through all selected statistics and plot a **histogram** to show how each stat is distributed across all players in the league.
    * `plt.hist(...)`: Draws the histogram with 20 bins and a blue color.
    * `safe_col`: Replaces "/" with "_" so the file can be saved without errors.
    * `plt.savefig(...)`: Saves each plot as a PNG image in high resolution.

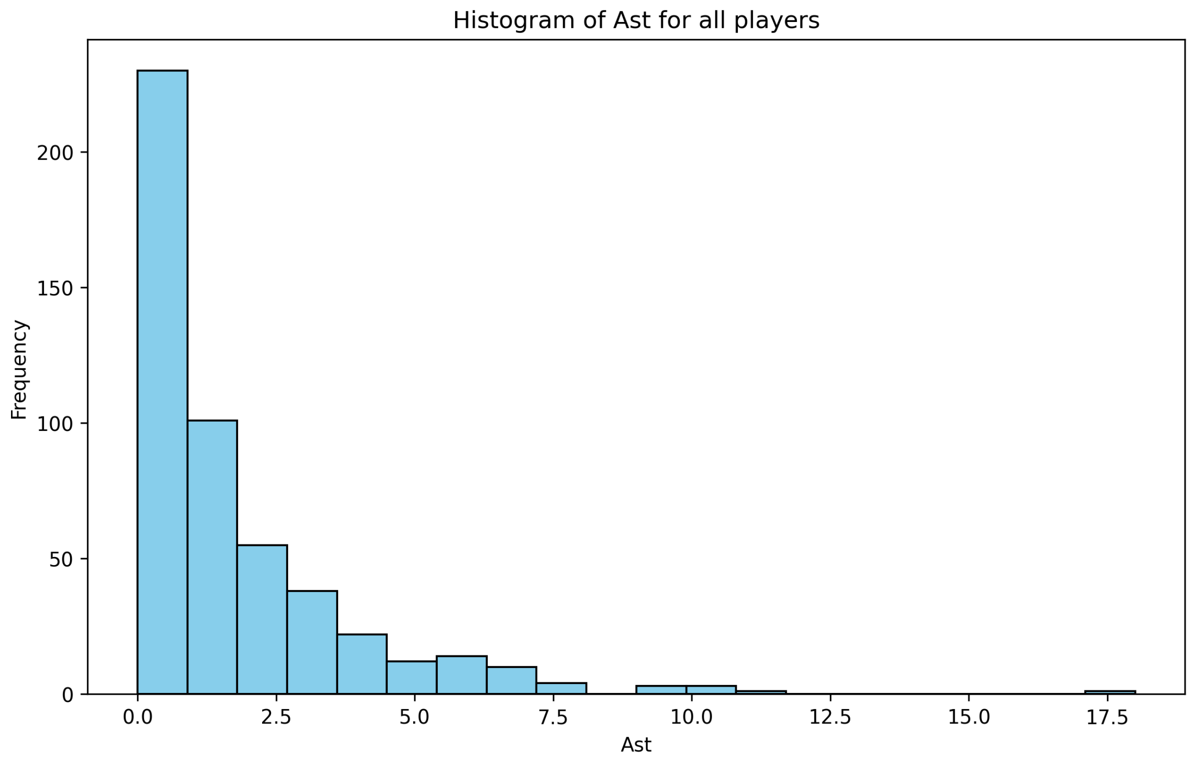
- **Plot Histograms for Each Team.**



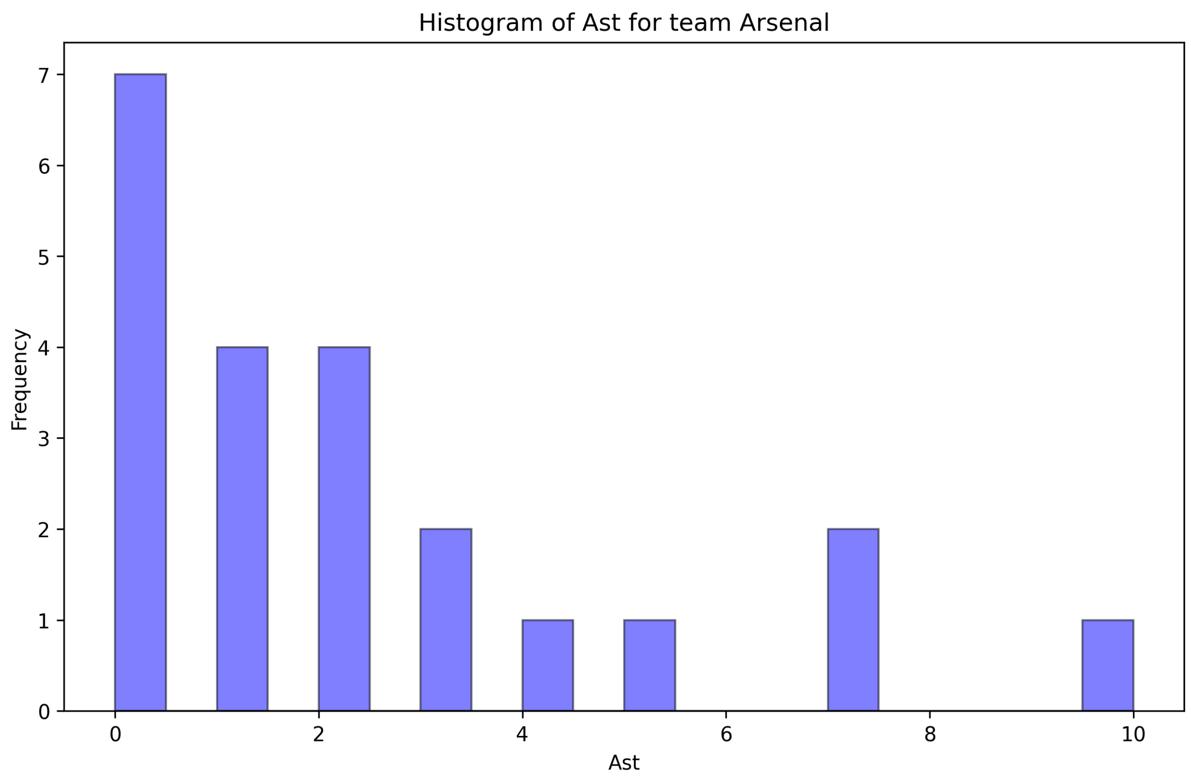
```python
# Histogram for each teams
for col in attack_cols + defense_cols :
    for team in df["Squad"].unique() : # Get unique values in the Squad column
        plt.figure(figsize = (10, 6))
        plt.hist(df[df["Squad"] == team][col], color = "blue", bins = 20, alpha = 0.5, edgecolor = "black")
        plt.title(f"Histogram of {col} for team {team}")
        plt.xlabel(col)
        plt.ylabel("Frequency")
        safe_col = col.replace("/", "_")  # Replace invalid characters in column names
        safe_team = team.replace("/", "_")  # Replace invalid characters in team names
        histogram_team_path = os.path.join("SourceCode", f"histogram_{safe_team}_{safe_col}.png")
        plt.savefig(histogram_team_path, dpi = 300, bbox_inches = "tight")
        plt.show()
```

  – This section repeats the histogram process for **each team separately**, so you can compare how stats like goals or tackles vary **within** a specific squad.
    * The loop for team in df["Squad"].unique() ensures that each team's histogram is created.
    * Saved filenames include both team and stat names, like histogram_Arsenal_Gls.png.

- **Example for Results.**
  - Histogram for all players :



– Histogram for each teams :



5. **Identify the team with the highest scores for each statistic.**

- **Remove the Overall League Row.**

```
table.drop(index = 0, inplace = True) # Remove the first row which is the overall stats for "all"
```

– Previously added a row summarizing overall league stats. Now have to remove it to only compare actual teams when identifying top performers.

- **Loop Over Each Statistic and Identify the Top Team.**

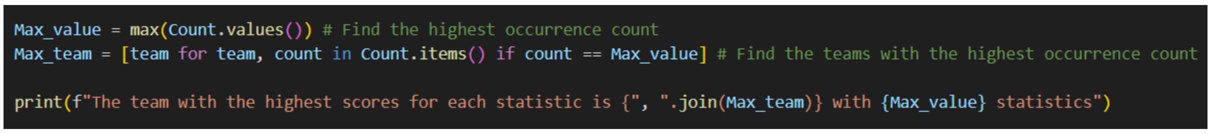
```
Count = {} # Count the occurrences of teams to print the best-performing team

for col in df.columns[4:] :
    mean_name = "Mean of " + col # Column name for mean
    # Find the team with the highest value using the index of the max value in the column
    max_team = table.loc[table[mean_name].idxmax(), "Squad"]
    max_value = table[mean_name].max() # Get the maximum value
    # Count occurrences
    if max_team not in Count :
        Count[max_team] = 1
    else :
        Count[max_team] += 1
    print(f"The team with the highest {mean_name} is {max_team} with a value of {max_value}")
```

– Loop through each player statistic (from column 4 onward).

– Use "Mean of " + col to match the correct column in the table DataFrame.

  - idxmax() finds the row index where the value is highest → we get the team name and max value.
  - We count how many times each team leads in any statistic using a dictionary Count.

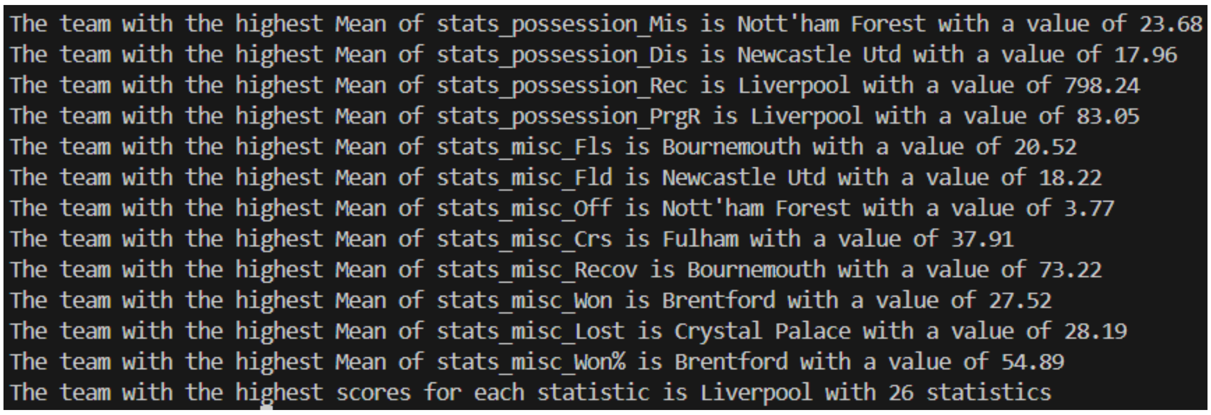
- **Identify the Overall Best-Performing Team.**



```python
Max_value = max(Count.values()) # Find the highest occurrence count
Max_team = [team for team, count in Count.items() if count == Max_value] # Find the teams with the highest occurrence count

print(f"The team with the highest scores for each statistic is {", ".join(Max_team)} with {Max_value} statistics")
```

  - After counting all top-performing teams, we find which team led the most often.
    * Max_value is the largest count.
    * Max_team contains all teams that reached this number. This helps identify the most consistently dominant team across all stats.

- **Example for Results.**



```
The team with the highest Mean of stats_possession_Mis is Nott'ham Forest with a value of 23.68
The team with the highest Mean of stats_possession_Dis is Newcastle Utd with a value of 17.96
The team with the highest Mean of stats_possession_Rec is Liverpool with a value of 798.24
The team with the highest Mean of stats_possession_PrgR is Liverpool with a value of 83.05
The team with the highest Mean of stats_misc_Fls is Bournemouth with a value of 20.52
The team with the highest Mean of stats_misc_Fld is Newcastle Utd with a value of 18.22
The team with the highest Mean of stats_misc_Off is Nott'ham Forest with a value of 3.77
The team with the highest Mean of stats_misc_Crs is Fulham with a value of 37.91
The team with the highest Mean of stats_misc_Recov is Bournemouth with a value of 73.22
The team with the highest Mean of stats_misc_Won is Brentford with a value of 27.52
The team with the highest Mean of stats_misc_Lost is Crystal Palace with a value of 28.19
The team with the highest Mean of stats_misc_Won% is Brentford with a value of 54.89
The team with the highest scores for each statistic is Liverpool with 26 statistics
```

## 2.4 4. Results

- The code successfully loads the data from `results.csv`
- It calculates and saves the top 3 highest and 3 lowest players for each statistic to `top_3.txt`
- It calculates and saves:
  - Median
  - Mean
  - Standard deviation

  for each statistic (all players and per team) to `results2.csv`
- It generates histograms visualizing the distribution of specified statistics
- It identifies and prints the team(s) with:
  - Most frequent highest scores
  - Best overall performance

# 3 PART III – PLAYER CLUSTERING

## 3.1 1. Assignment Requirements

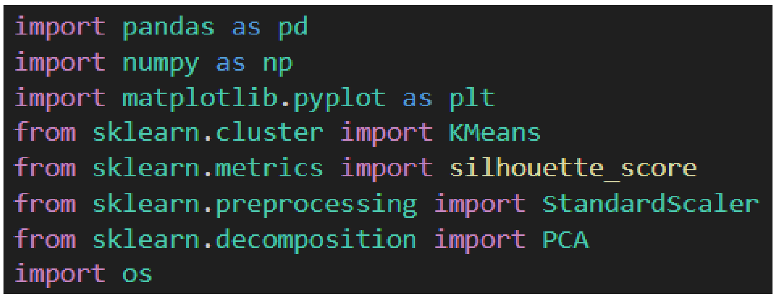
- Use the K-means algorithm to classify players into groups based on their statistics.
- How many groups should the players be classified into? Why? Provide your comments on the results.
- Use PCA to reduce the data dimensions to 2, then plot a 2D cluster of the data points.

## 3.2 2. Assignment Breakdown

1. **Data Loading and Preparation:** Load the cleaned data and prepare it for clustering.
2. **K-Means Clustering:** Apply the K-means algorithm to group players.
3. **Determine Optimal Clusters:** Determine the optimal number of clusters (k).
4. **Dimensionality Reduction:** Use PCA to reduce the data to 2 dimensions.
5. **Cluster Visualization:** Plot the clusters in a 2D scatter plot.
6. **Result Interpretation:** Comment on the characteristics of the clusters.

## 3.3 3. Solution Approach

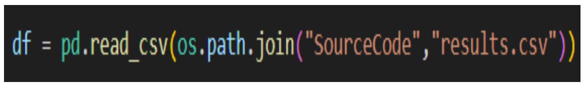
1. **Data Preparation.**
   - **Import necessary libraries.**



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import os
```

     – `pandas`: Handles and manipulates data in tabular form (e.g., filtering, cleaning, merging).
     – `numpy`: Provides support for large arrays and matrices, along with mathematical functions.
     – `matplotlib.pyplot`: Used for creating visualizations like plots and charts.
     – `sklearn.cluster.KMeans`: Performs K-Means clustering to group data into clusters.
     – `sklearn.metrics.silhouette_score`: Measures how well-defined the clusters are.
     – `sklearn.preprocessing.StandardScaler`: Scales data to have a mean of 0 and a standard deviation of 1.
     – `sklearn.decomposition.PCA`: Reduces dimensionality of data for easier visualization and analysis.
     – os: Interact with the operating system, check for file paths, create folders, and manage directories.

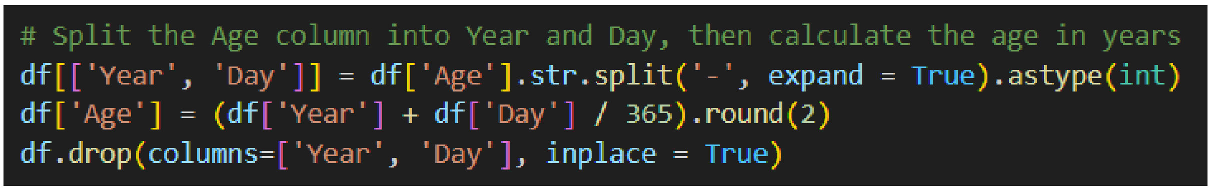
- **Read and Preprocess Data.**



```
df = pd.read_csv(os.path.join("SourceCode","results.csv"))
```

  – Reads a CSV file named results.csv into a DataFrame called df.

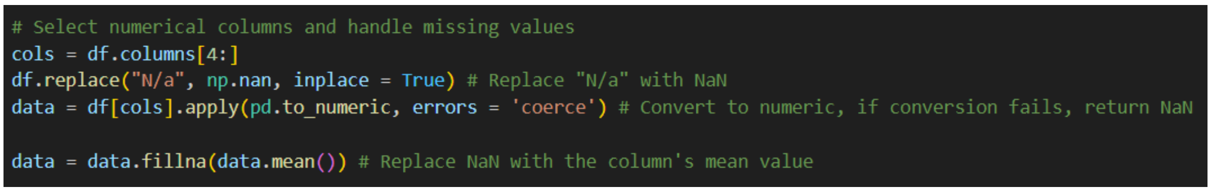
- **Preprocess Age Data.**



```
# Split the Age column into Year and Day, then calculate the age in years
df[['Year', 'Day']] = df['Age'].str.split('-', expand = True).astype(int)
df['Age'] = (df['Year'] + df['Day'] / 365).round(2)
df.drop(columns=['Year', 'Day'], inplace = True)
```

  – The Age column is initially in the format XX-YY (e.g. 24-192 meaning 24 years and 192 days).
  – This code splits it into two columns: Year and Day.
  – Then it calculates the exact age in years with decimal and replaces the original Age column.

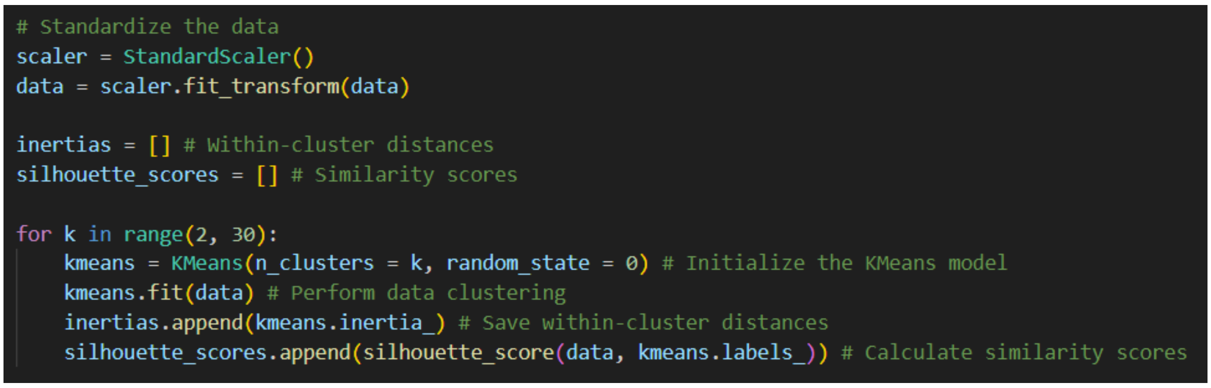
- **Handle Missing and Non-Numeric Data.**



```
# Select numerical columns and handle missing values
cols = df.columns[4:]
df.replace("N/a", np.nan, inplace = True) # Replace "N/a" with NaN
data = df[cols].apply(pd.to_numeric, errors = 'coerce') # Convert to numeric, if conversion fails, return NaN

data = data.fillna(data.mean()) # Replace NaN with the column's mean value
```

  – Selects all columns starting from the 5th column onward (excluding Player, Nation, Squad, Pos).
  – Replaces any ”N/a” entries with NaN using np.nan.
  – Converts all selected columns to numeric values; if conversion fails, values become NaN.
  – Replaces all missing values (NaN) with the mean of each respective column.

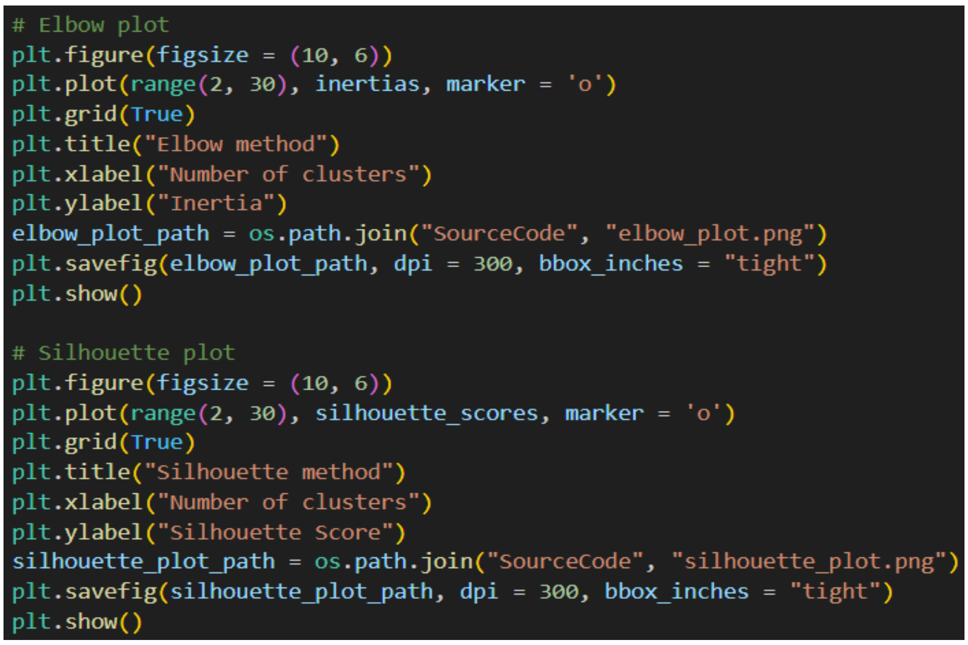
2. **Standardize Data & Evaluate Optimal Cluster Count.**



```python
# Standardize the data
scaler = StandardScaler()
data = scaler.fit_transform(data)

inertias = [] # Within-cluster distances
silhouette_scores = [] # Similarity scores

for k in range(2, 30):
    kmeans = KMeans(n_clusters = k, random_state = 0) # Initialize the KMeans model
    kmeans.fit(data) # Perform data clustering
    inertias.append(kmeans.inertia_) # Save within-cluster distances
    silhouette_scores.append(silhouette_score(data, kmeans.labels_)) # Calculate similarity scores
```

- The data is standardized using StandardScaler() to ensure all features are on the same scale.
- A loop from k = 2 to 29 runs KMeans clustering for each value of k.
- For each clustering, the following are stored:
  - Inertia: how tight the clusters are.
  - Silhouette score: how well-separated the clusters are.

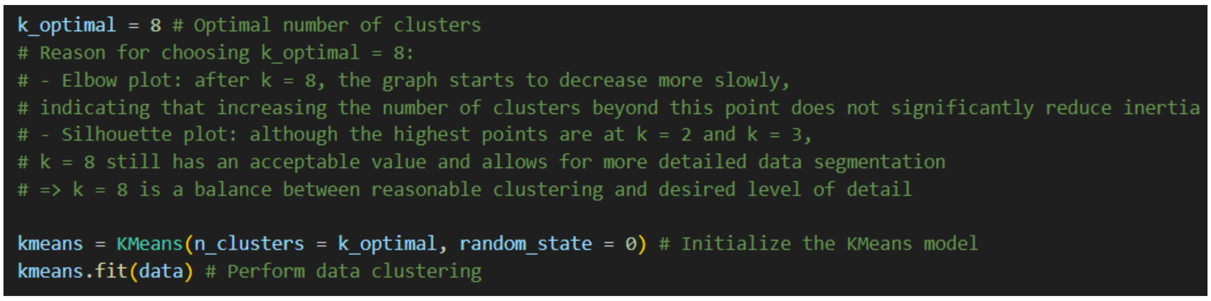
3. **Visualize Elbow & Silhouette Methods.**



```python
# Elbow plot
plt.figure(figsize = (10, 6))
plt.plot(range(2, 30), inertias, marker = 'o')
plt.grid(True)
plt.title("Elbow method")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
elbow_plot_path = os.path.join("SourceCode", "elbow_plot.png")
plt.savefig(elbow_plot_path, dpi = 300, bbox_inches = "tight")
plt.show()

# Silhouette plot
plt.figure(figsize = (10, 6))
plt.plot(range(2, 30), silhouette_scores, marker = 'o')
plt.grid(True)
plt.title("Silhouette method")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette Score")
silhouette_plot_path = os.path.join("SourceCode", "silhouette_plot.png")
plt.savefig(silhouette_plot_path, dpi = 300, bbox_inches = "tight")
plt.show()
```

- Two plots help evaluate the optimal value of k:
  - Elbow Method: Looks for the point where inertia stops decreasing sharply.
  - Silhouette Method: Chooses k with high separation between clusters.

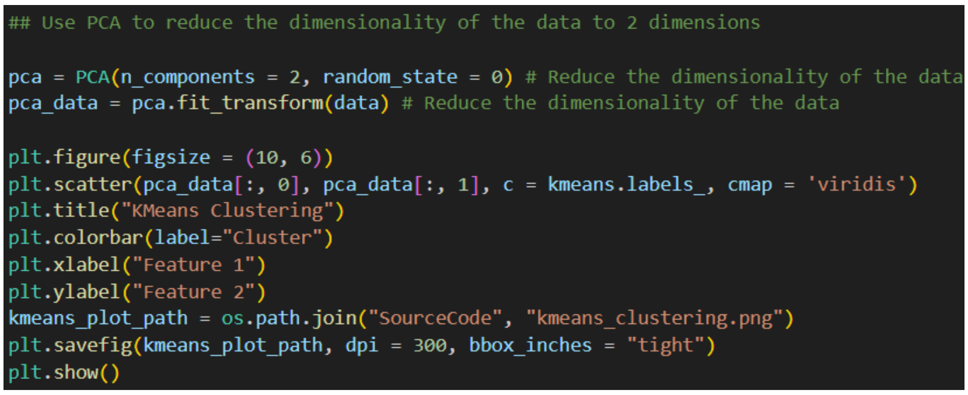
4. **Apply Clustering with Chosen k = 8**



```python
k_optimal = 8 # Optimal number of clusters
# Reason for choosing k_optimal = 8:
# - Elbow plot: after k = 8, the graph starts to decrease more slowly,
# indicating that increasing the number of clusters beyond this point does not significantly reduce inertia
# - Silhouette plot: although the highest points are at k = 2 and k = 3,
# k = 8 still has an acceptable value and allows for more detailed data segmentation
# => k = 8 is a balance between reasonable clustering and desired level of detail

kmeans = KMeans(n_clusters = k_optimal, random_state = 0) # Initialize the KMeans model
kmeans.fit(data) # Perform data clustering
```

- k = 8 is chosen because:
  - In the elbow plot, the drop in inertia slows after k = 8.
  - In the silhouette plot, score is still good at k = 8.
  - It balances clustering quality and cluster detail level.

5. **Visualize Clustering Results Using PCA (2D Projection).**

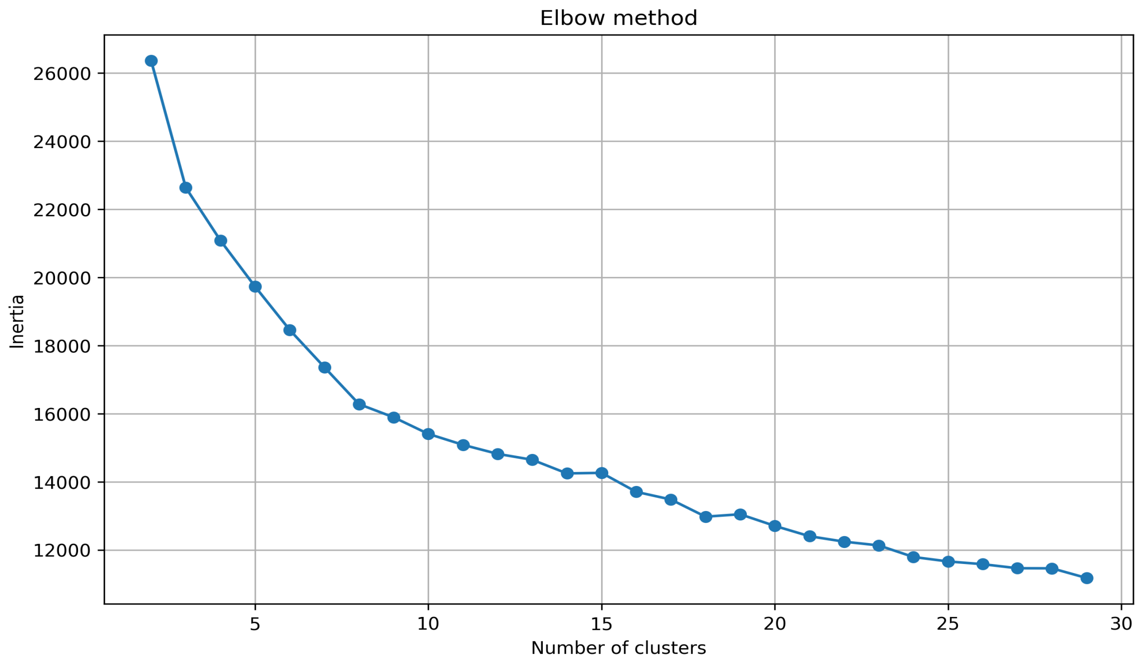


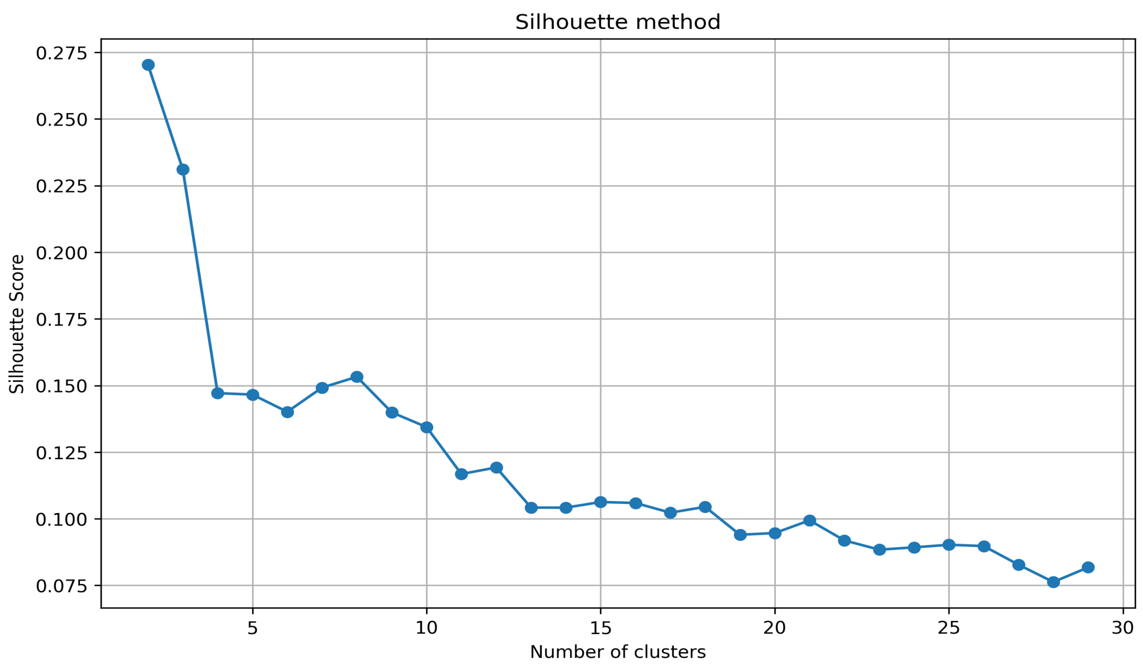
```python
## Use PCA to reduce the dimensionality of the data to 2 dimensions

pca = PCA(n_components = 2, random_state = 0) # Reduce the dimensionality of the data
pca_data = pca.fit_transform(data) # Reduce the dimensionality of the data

plt.figure(figsize = (10, 6))
plt.scatter(pca_data[:, 0], pca_data[:, 1], c = kmeans.labels_, cmap = 'viridis')
plt.title("KMeans Clustering")
plt.colorbar(label="Cluster")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
kmeans_plot_path = os.path.join("SourceCode", "kmeans_clustering.png")
plt.savefig(kmeans_plot_path, dpi = 300, bbox_inches = "tight")
plt.show()
```

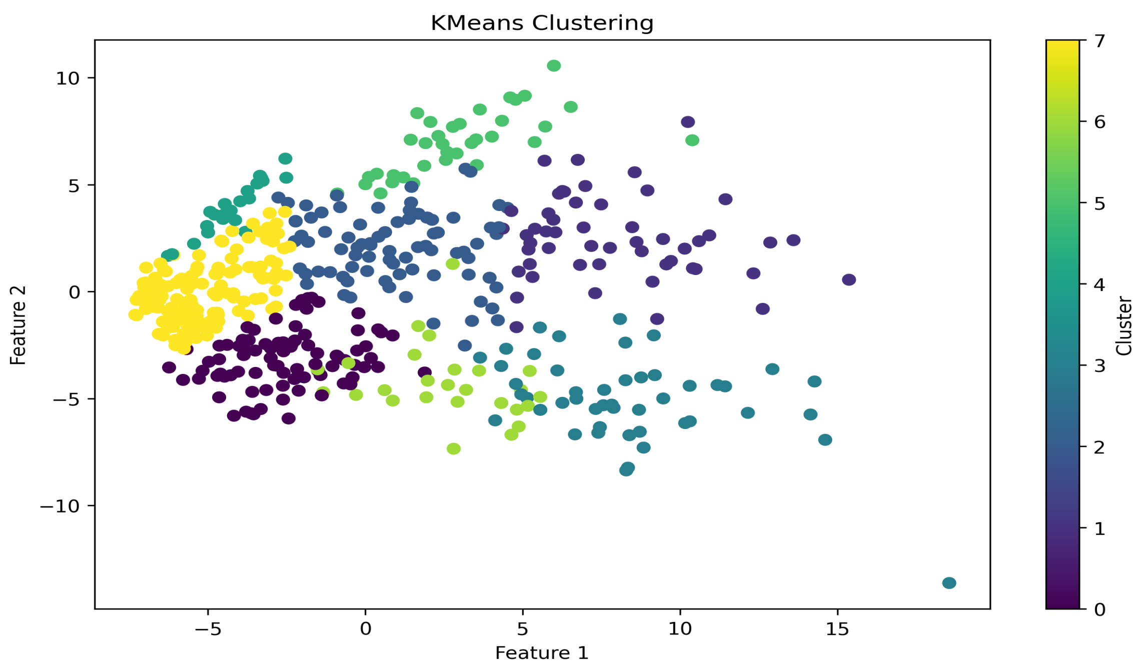
- PCA (Principal Component Analysis) reduces the dataset to 2 main components to visualize complex, high-dimensional data in a 2D plot.
- Each point in the scatter plot represents a player, colored by its assigned cluster.
- This helps to visually assess how well the KMeans algorithm separated the data into clusters.

## 3.4 4. Results

- The code successfully loads and preprocesses the data.
- It performs K-means clustering for various numbers of clusters.
- It generates elbow and silhouette plots to aid in determining the optimal number of clusters.
- It chooses k = 8 as the optimal number of clusters and provides a justification.
- It reduces the data to 2 dimensions using PCA.
- It visualizes the clusters in a 2D scatter plot.
- Example :




Silhouette method
Silhouette Score
Number of clusters
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
5
10
15
20
25
30


KMeans Clustering
Feature 2
Feature 1
Cluster
10
5
0
−5
−10
−5
0
5
10
15
0
1
2
3
4
5
6
7

# 4 PART IV – PLAYER VALUE ESTIMATION

## 4.1 1. Assignment Requirements

- Collect player transfer values for the 2024-2025 season from `https://www.footballtransfers.com`. Note that only collect for the players whose playing time is greater than 900 minutes
- Propose a method for estimating player values. How do you select feature and model?

## 4.2 2. Assignment Breakdown

**Part 4.1: Collect player transfer values for the 2024-2025 season, only collect for the players whose playing time is greater than 900 minutes.**

- Web Scraping: Extract player transfer values from the specified website.
- Data Filtering: Filter players based on the 900-minute threshold.
- Data Merging: Combine the transfer value data with the player statistics.

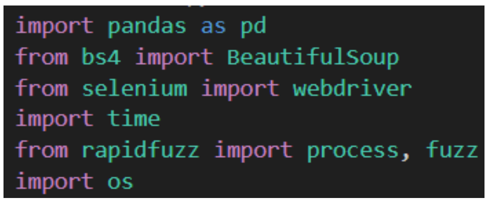
**Part 4.2: Estimating Player Transfer Values Using Linear Regression.**

- Feature Selection: Choose relevant features for the value estimation model.
- Model Selection: Select an appropriate model for estimating player values.
- Model Training: Train the chosen model on the data.
- Model Evaluation: Evaluate the performance of the model.

## 4.3 3. Solution Approach

**Part 4.1 : Collect player transfer values for the 2024-2025 season, only collect for the players whose playing time is greater than 900 minutes.**

1. **Import necessary libraries.**

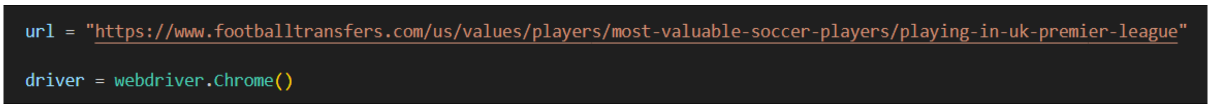


```python
import pandas as pd
from bs4 import BeautifulSoup
from selenium import webdriver
import time
from rapidfuzz import process, fuzz
import os
```

- pandas: Used for storing and manipulating structured data (especially DataFrames).
- BeautifulSoup: Parses and extracts information from HTML content.
- selenium: Automates web browsing (used to open pages and get dynamic HTML).
- time: Adds delays to ensure pages load properly.

- rapidfuzz: Performs fuzzy string matching to handle inconsistent player name formats.
- os: Interact with the operating system, check for file paths, create folders, and manage directories.

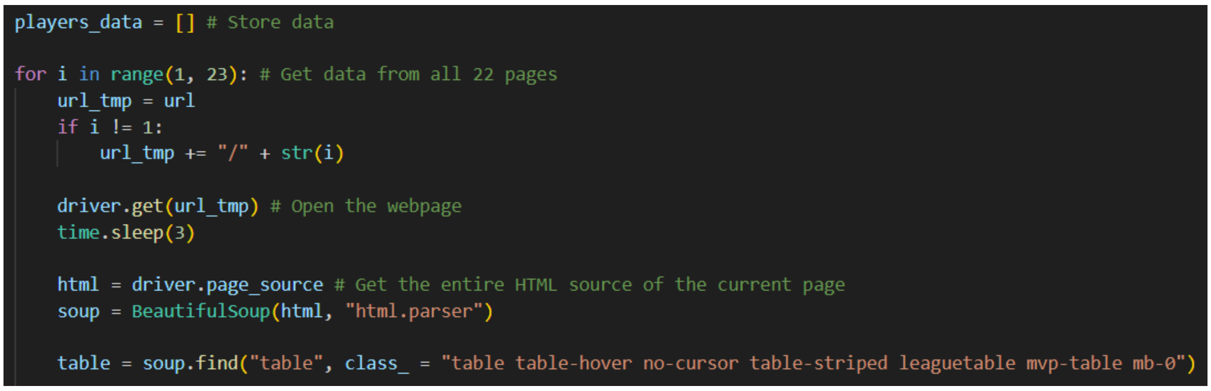
2. **Define the URL and Launch the Browser.**



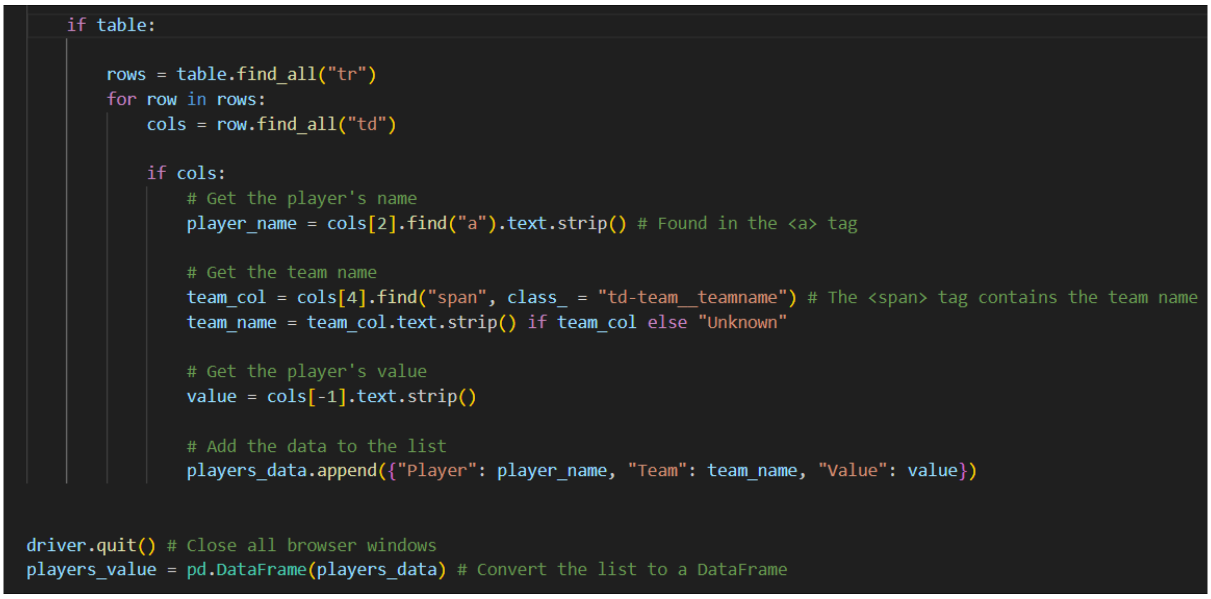
```
url = "https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league"

driver = webdriver.Chrome()
```

- Define the base URL where the player value data is located and initialize the Selenium Chrome browser.

3. **Scrape Premier League Player Transfer Values.**



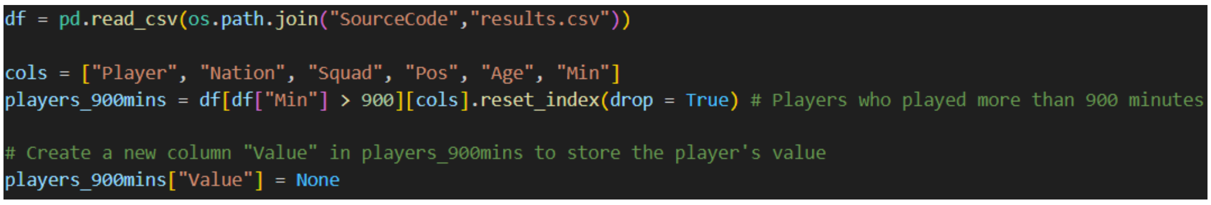
```
players_data = [] # Store data

for i in range(1, 23): # Get data from all 22 pages
    url_tmp = url
    if i != 1:
        url_tmp += "/" + str(i)

    driver.get(url_tmp) # Open the webpage
    time.sleep(3)

    html = driver.page_source # Get the entire HTML source of the current page
    soup = BeautifulSoup(html, "html.parser")

    table = soup.find("table", class_ = "table table-hover no-cursor table-striped leaguetable mvp-table mb-0")
```



```
    if table:

        rows = table.find_all("tr")
        for row in rows:
            cols = row.find_all("td")

            if cols:
                # Get the player's name
                player_name = cols[2].find("a").text.strip() # Found in the <a> tag

                # Get the team name
                team_col = cols[4].find("span", class_ = "td-team__teamname") # The <span> tag contains the team name
                team_name = team_col.text.strip() if team_col else "Unknown"

                # Get the player's value
                value = cols[-1].text.strip()

                # Add the data to the list
                players_data.append({"Player": player_name, "Team": team_name, "Value": value})


driver.quit() # Close all browser windows
players_value = pd.DataFrame(players_data) # Convert the list to a DataFrame
```

- Use Selenium and BeautifulSoup to scrape transfer value data from all 22 pages on Football-Transfers.com.
- For each page, it loads the HTML, locates the target table, and extracts each player's name, club, and estimated value.
- The data is collected into a list of dictionaries (players_data), then converted into a pandas DataFrame (players_value).

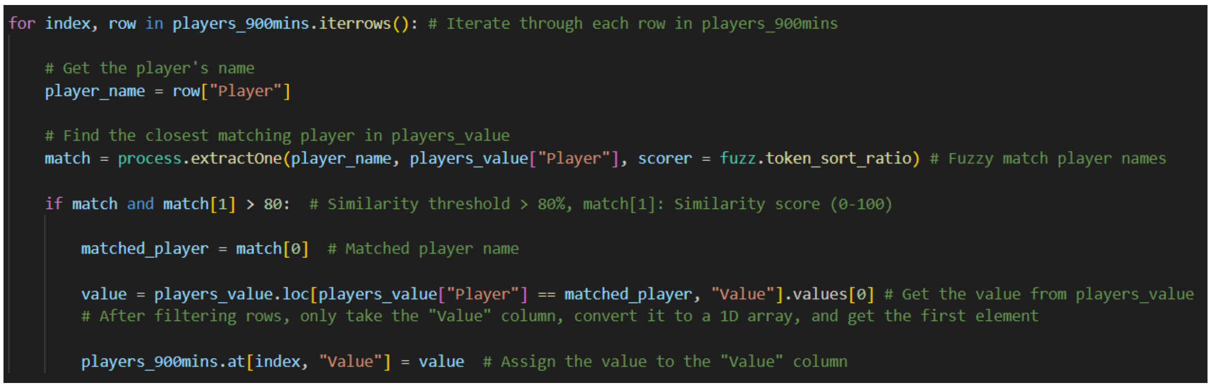
4. **Load Data and Filter Players with Over 900 Minutes.**



```python
df = pd.read_csv(os.path.join("SourceCode","results.csv"))

cols = ["Player", "Nation", "Squad", "Pos", "Age", "Min"]
players_900mins = df[df["Min"] > 900][cols].reset_index(drop = True) # Players who played more than 900 minutes

# Create a new column "Value" in players_900mins to store the player's value
players_900mins["Value"] = None
```

- Load the player match statistics from the CSV file results.csv.
- Filter only players who have played more than 900 minutes. Keep only selected columns, then reset the index of the filtered DataFrame.
- Add a new column ”Value” to the filtered DataFrame to later store the player’s estimated transfer value.

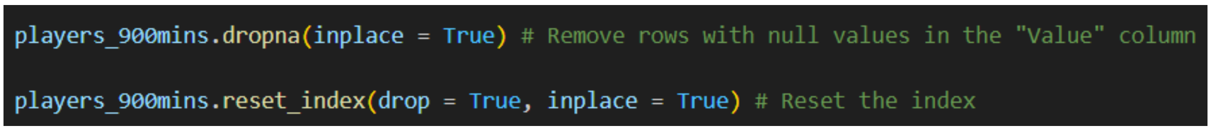
5. **Match Player Names Using Fuzzy Matching.**



```python
for index, row in players_900mins.iterrows(): # Iterate through each row in players_900mins

    # Get the player's name
    player_name = row["Player"]

    # Find the closest matching player in players_value
    match = process.extractOne(player_name, players_value["Player"], scorer = fuzz.token_sort_ratio) # Fuzzy match player names

    if match and match[1] > 80:  # Similarity threshold > 80%, match[1]: Similarity score (0-100)

        matched_player = match[0]  # Matched player name

        value = players_value.loc[players_value["Player"] == matched_player, "Value"].values[0] # Get the value from players_value
        # After filtering rows, only take the "Value" column, convert it to a 1D array, and get the first element

        players_900mins.at[index, "Value"] = value  # Assign the value to the "Value" column
```

- Loop through each player in the filtered list and use fuzzy string matching to find the closest name in the scraped transfer values. If the match score is above 80, assign the matched value to the ”Value” column.

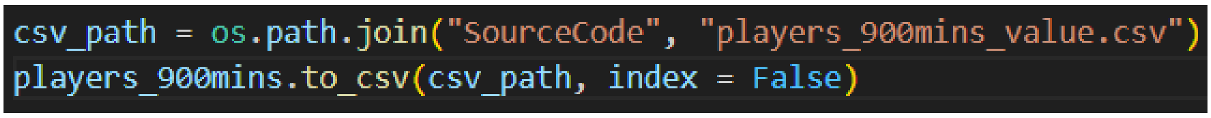
6. **Remove Unmatched Players and Finalize.**



```python
players_900mins.dropna(inplace = True) # Remove rows with null values in the "Value" column

players_900mins.reset_index(drop = True, inplace = True) # Reset the index
```

- Remove any players who couldn’t be matched (i.e. have NaN in the ”Value” column), then reset the index.

7. **Save the Final Data to CSV.**

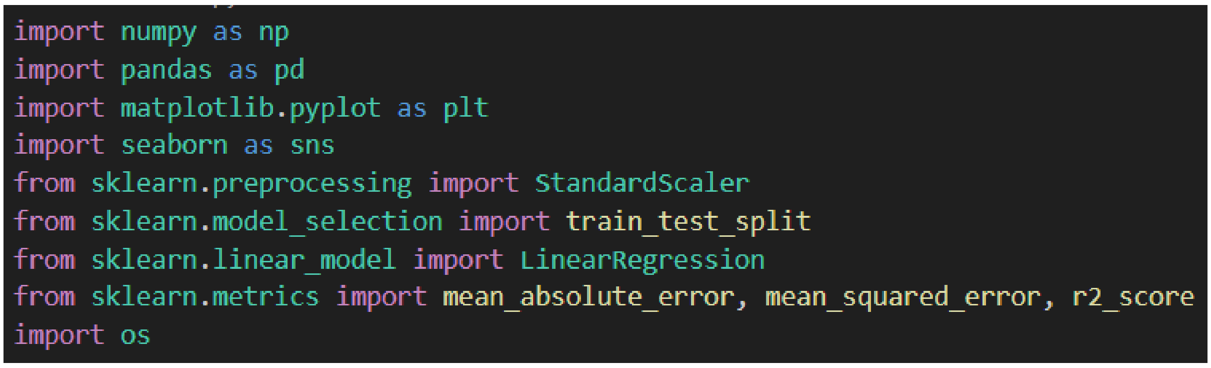


```
csv_path = os.path.join("SourceCode", "players_900mins_value.csv")
players_900mins.to_csv(csv_path, index = False)
```

- Export the final cleaned dataset (players with over 900 minutes and matched values) to a new CSV file.

**Part 4.2 : Estimating Player Transfer Values Using Linear Regression.**

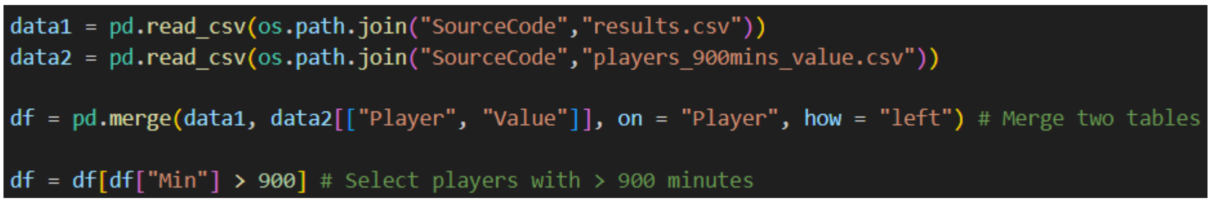
1. **Import necessary libraries.**



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import os
```

- pandas: Used for storing and manipulating structured data (especially DataFrames).
- numpy: Provides support for large arrays and matrices, along with mathematical functions.
- matplotlib: A plotting library used to create static, animated, and interactive visualizations.
- seaborn: Built on top of Matplotlib, it provides a high-level interface for creating statistical data visualizations.
- StandardScaler (from sklearn.preprocessing): Scales features by removing the mean and scaling to unit variance, improving model performance.
- train_test_split (from sklearn.model_selection): Splits datasets into training and testing subsets for model evaluation.
- LinearRegression (from sklearn.linear_model): A machine learning model for predicting a dependent variable using a linear relationship with one or more independent variables.
- mean_absolute_error, mean_squared_error, r2_score (from sklearn.metrics): Metrics to evaluate regression model performance, measuring prediction accuracy and model fit.
- os: Interact with the operating system, check for file paths, create folders, and manage directories.

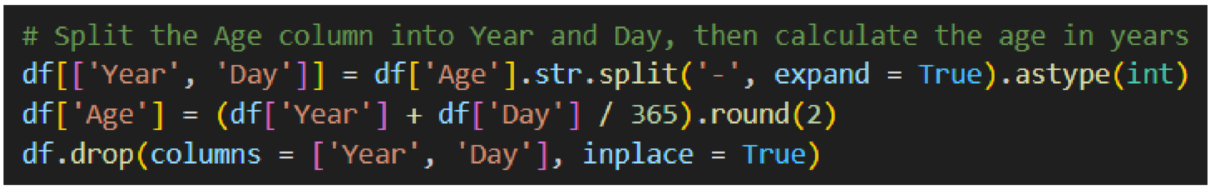
2. **Data Merging & Cleaning.**



```python
data1 = pd.read_csv(os.path.join("SourceCode","results.csv"))
data2 = pd.read_csv(os.path.join("SourceCode","players_900mins_value.csv"))

df = pd.merge(data1, data2[["Player", "Value"]], on = "Player", how = "left") # Merge two tables

df = df[df["Min"] > 900] # Select players with > 900 minutes
```

- data1 and data2 are merged based on the ”Player” column to form a combined dataframe (df).
- Players who have more than 900 minutes of playtime are selected for analysis.

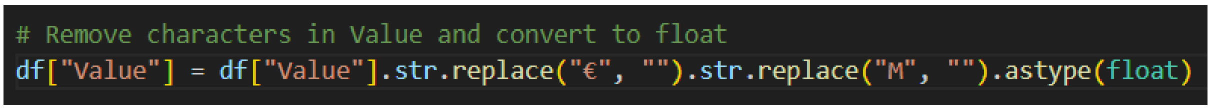
3. **Preprocess Age Data.**



```python
# Split the Age column into Year and Day, then calculate the age in years
df[['Year', 'Day']] = df['Age'].str.split('-', expand = True).astype(int)
df['Age'] = (df['Year'] + df['Day'] / 365).round(2)
df.drop(columns = ['Year', 'Day'], inplace = True)
```

- The ”Age” column, originally in the format of years-days, is split into separate ”Year” and ”Day” columns, then used to calculate the player’s age in years (with fractional days).

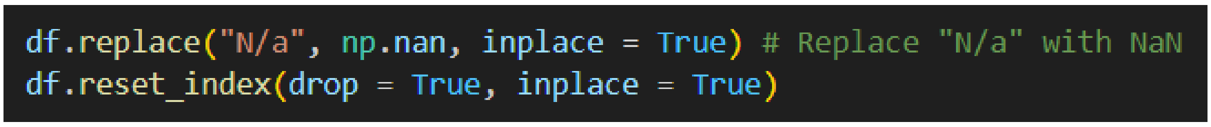
4. **Preprocess Value Data.**



```python
# Remove characters in Value and convert to float
df["Value"] = df["Value"].str.replace("€", "").str.replace("M", "").astype(float)
```

- The player value (column ”Value”) is cleaned by removing characters like ”€” and ”M”, and then converted to float.

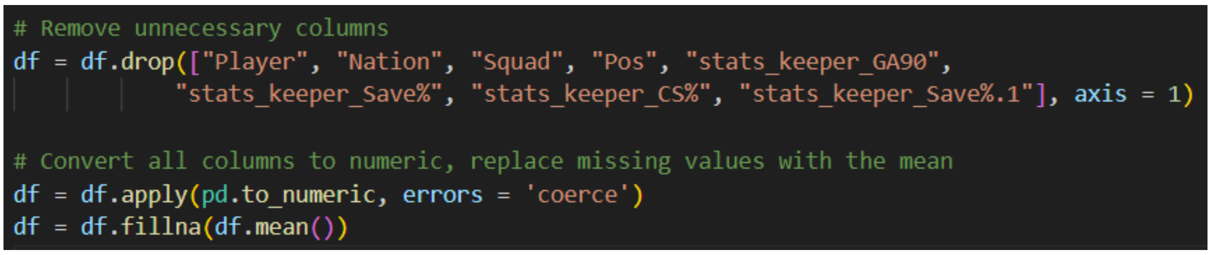
5. **Replace Missing Values and Reset Index.**



```
df.replace("N/a", np.nan, inplace = True) # Replace "N/a" with NaN
df.reset_index(drop = True, inplace = True)
```

- Missing values marked as ”N/a” are replaced with NaN for easier processing. Index is reset.

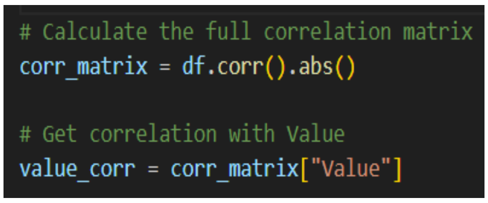
6. **Feature Engineering.**



```
# Remove unnecessary columns
df = df.drop(["Player", "Nation", "Squad", "Pos", "stats_keeper_GA90",
            "stats_keeper_Save%", "stats_keeper_CS%", "stats_keeper_Save%.1"], axis = 1)

# Convert all columns to numeric, replace missing values with the mean
df = df.apply(pd.to_numeric, errors = 'coerce')
df = df.fillna(df.mean())
```

- Irrelevant columns and goalkeeper-specific stats are dropped.
- All columns are converted to numeric type. Missing values are filled with the column mean.

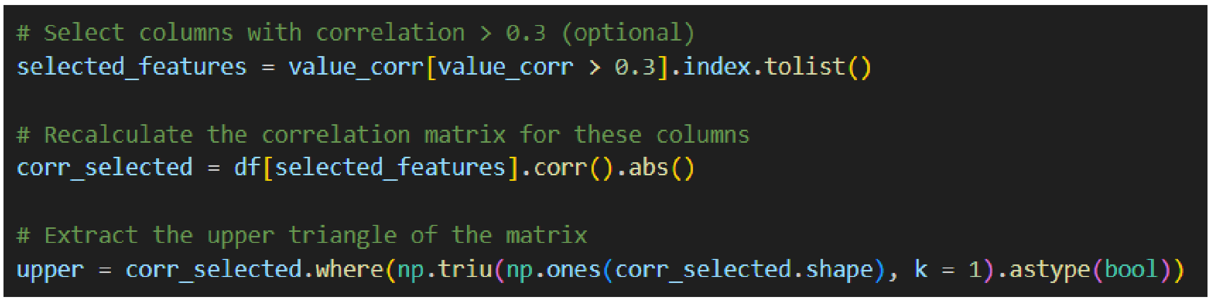
7. **Calculate Correlation With Target.**



```
# Calculate the full correlation matrix
corr_matrix = df.corr().abs()

# Get correlation with Value
value_corr = corr_matrix["Value"]
```

- The correlation matrix is computed and the correlation of each feature with the target (Value) is extracted.

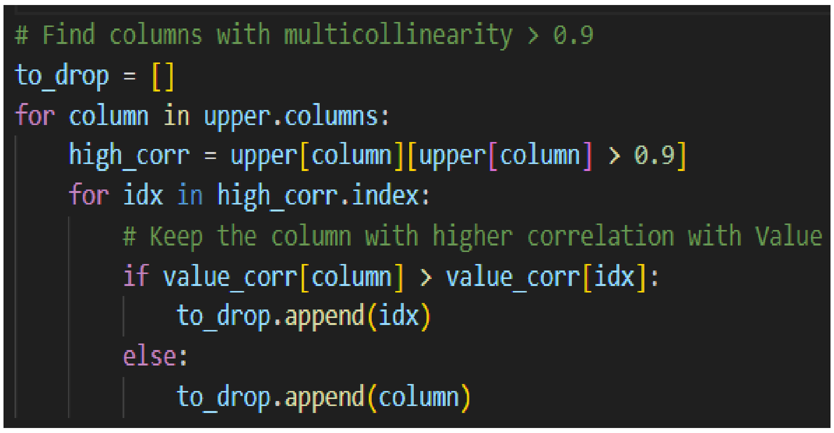
8. **Select Features With Correlation > 0.3, Identify Multicollinear Feature Pairs.**



```
# Select columns with correlation > 0.3 (optional)
selected_features = value_corr[value_corr > 0.3].index.tolist()

# Recalculate the correlation matrix for these columns
corr_selected = df[selected_features].corr().abs()

# Extract the upper triangle of the matrix
upper = corr_selected.where(np.triu(np.ones(corr_selected.shape), k = 1).astype(bool))
```

- Features with correlation $> 0.3$ with Value are selected for modeling.
- Upper triangle of the correlation matrix is used to detect highly correlated feature pairs.

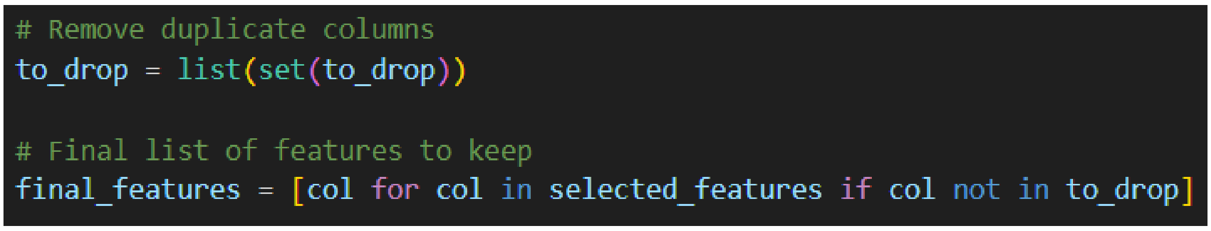
9. **Drop One Feature From Each Highly Correlated Pair.**



```
# Find columns with multicollinearity > 0.9
to_drop = []
for column in upper.columns:
    high_corr = upper[column][upper[column] > 0.9]
    for idx in high_corr.index:
        # Keep the column with higher correlation with Value
        if value_corr[column] > value_corr[idx]:
            to_drop.append(idx)
        else:
            to_drop.append(column)
```

- In each highly correlated pair (correlation $> 0.9$), the feature less correlated with Value is dropped.

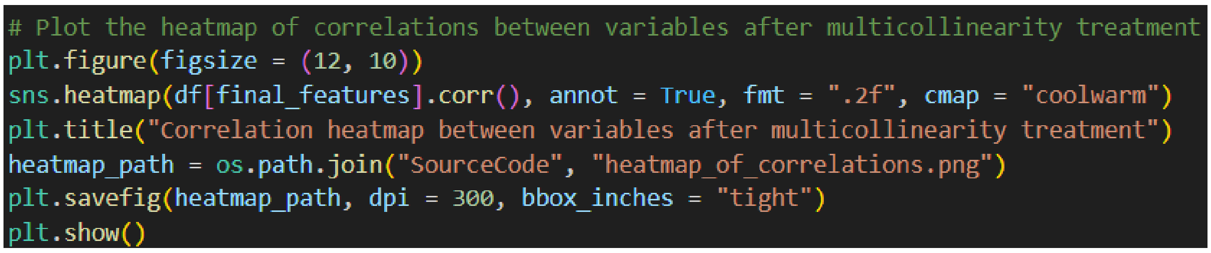
10. **Remove Duplicate Dropped Features and Final List of Features.**



```
# Remove duplicate columns
to_drop = list(set(to_drop))

# Final list of features to keep
final_features = [col for col in selected_features if col not in to_drop]
```

- Removes duplicate entries from the list of features to drop.
- Gets the final list of selected features after removing multicollinear ones.

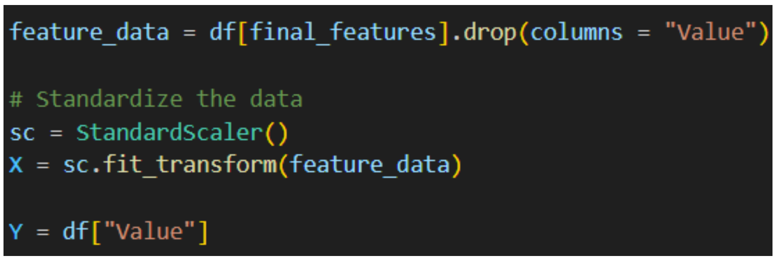
11. **Heatmap of Final Feature Correlation.**



```python
# Plot the heatmap of correlations between variables after multicollinearity treatment
plt.figure(figsize = (12, 10))
sns.heatmap(df[final_features].corr(), annot = True, fmt = ".2f", cmap = "coolwarm")
plt.title("Correlation heatmap between variables after multicollinearity treatment")
heatmap_path = os.path.join("SourceCode", "heatmap_of_correlations.png")
plt.savefig(heatmap_path, dpi = 300, bbox_inches = "tight")
plt.show()
```

- A heatmap is drawn to visualize the correlations between the remaining features.

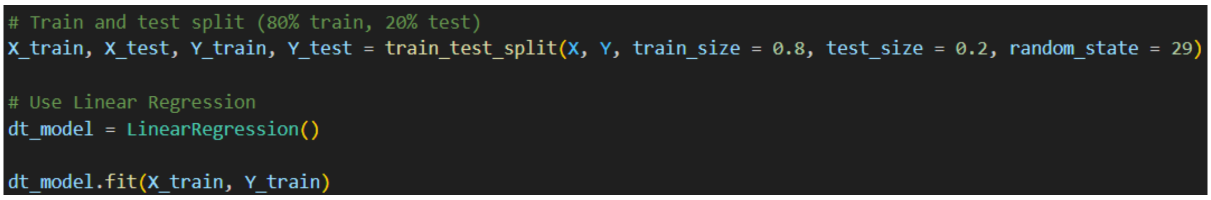
12. **Standardize Features and Define Target.**



```python
feature_data = df[final_features].drop(columns = "Value")

# Standardize the data
sc = StandardScaler()
X = sc.fit_transform(feature_data)

Y = df["Value"]
```

- Features are scaled to have zero mean and unit variance to improve model performance.

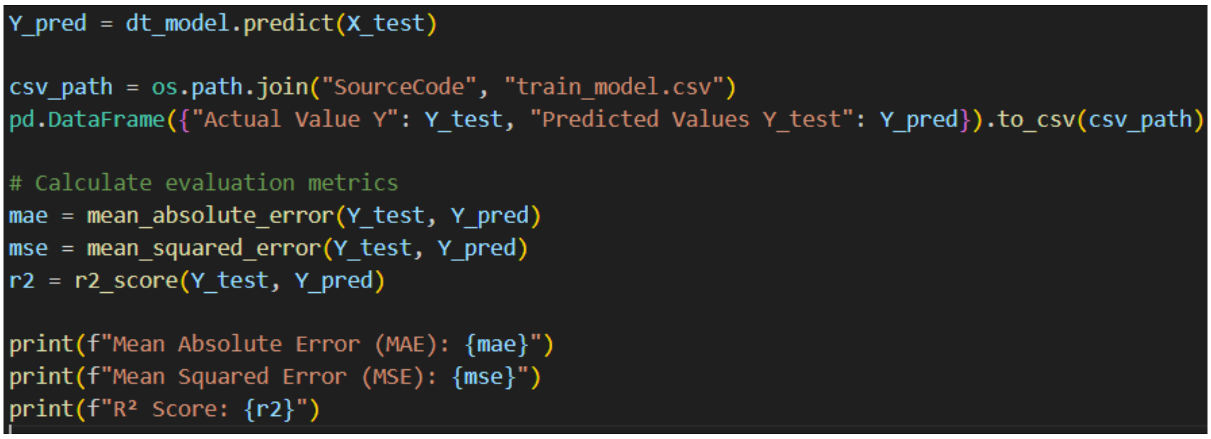
13. **Train-Test Split, Train Linear Regression Model.**



```python
# Train and test split (80% train, 20% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.8, test_size = 0.2, random_state = 29)

# Use Linear Regression
dt_model = LinearRegression()

dt_model.fit(X_train, Y_train)
```

- Splits the data into 80% training and 20% testing sets.
- Fits the linear regression model to training data.

14. **Evaluation and Output.**



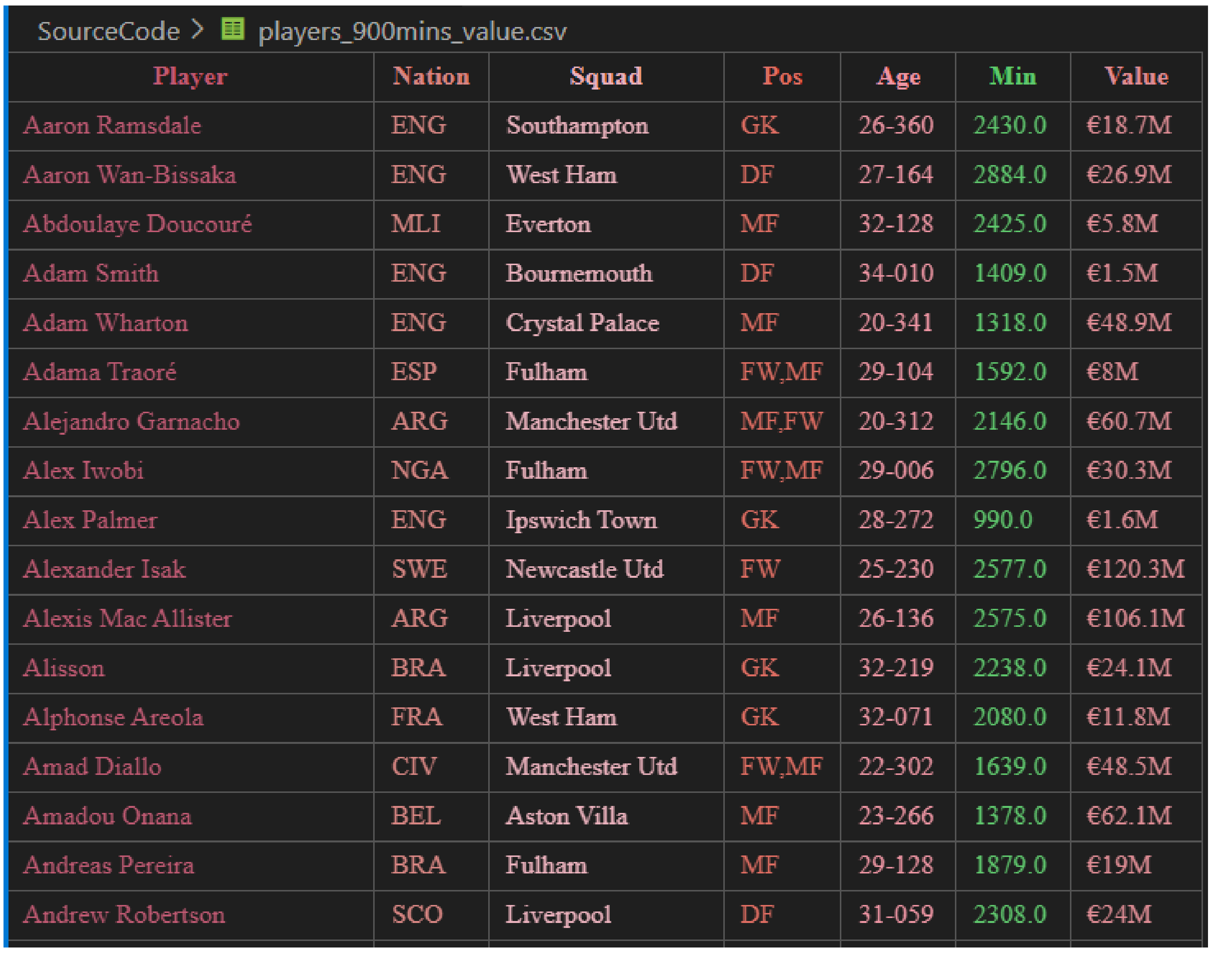
```
Y_pred = dt_model.predict(X_test)

csv_path = os.path.join("SourceCode", "train_model.csv")
pd.DataFrame({"Actual Value Y": Y_test, "Predicted Values Y_test": Y_pred}).to_csv(csv_path)

# Calculate evaluation metrics
mae = mean_absolute_error(Y_test, Y_pred)
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

- Predictions on the test set are saved along with actual values into a CSV file.
- Calculates MAE, MSE, and $R^2$ metrics to evaluate model performance.
  - **MAE**: Average error magnitude (e.g. €12.07M)
  - **MSE**: Squared error (more sensitive to large errors)
  - **$R^2$ Score**: Proportion of variance explained by the model (e.g. $0.615 = 61.5$
- Displays how well the model predicts player values.

## 4.4 4. Results

**Part 4.1: Collect player transfer values for the 2024-2025 season, only collect for the players whose playing time is greater than 900 minutes.**

- The code successfully scrapes player transfer values.
- It merges this data with player statistics, handling potential name discrepancies with fuzzy matching.
- It saves the merged data to a CSV file.
- Example :

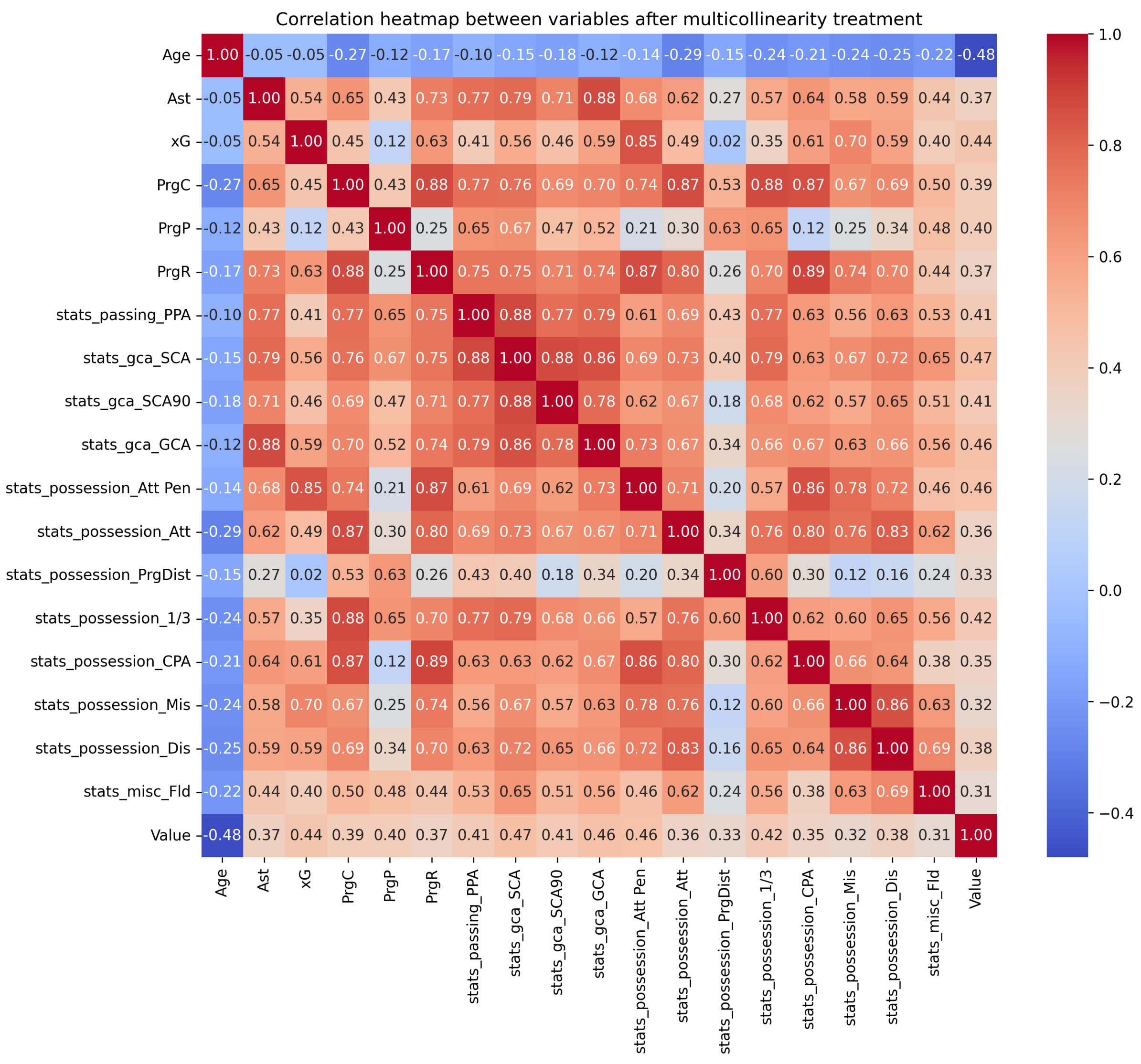


SourceCode > players_900mins_value.csv

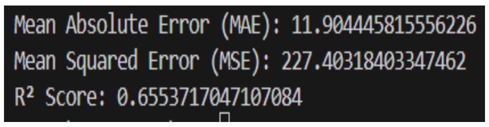
| Player | Nation | Squad | Pos | Age | Min | Value |
|---|---|---|---|---|---|---|
| Aaron Ramsdale | ENG | Southampton | GK | 26-360 | 2430.0 | €18.7M |
| Aaron Wan-Bissaka | ENG | West Ham | DF | 27-164 | 2884.0 | €26.9M |
| Abdoulaye Doucouré | MLI | Everton | MF | 32-128 | 2425.0 | €5.8M |
| Adam Smith | ENG | Bournemouth | DF | 34-010 | 1409.0 | €1.5M |
| Adam Wharton | ENG | Crystal Palace | MF | 20-341 | 1318.0 | €48.9M |
| Adama Traoré | ESP | Fulham | FW,MF | 29-104 | 1592.0 | €8M |
| Alejandro Garnacho | ARG | Manchester Utd | MF,FW | 20-312 | 2146.0 | €60.7M |
| Alex Iwobi | NGA | Fulham | FW,MF | 29-006 | 2796.0 | €30.3M |
| Alex Palmer | ENG | Ipswich Town | GK | 28-272 | 990.0 | €1.6M |
| Alexander Isak | SWE | Newcastle Utd | FW | 25-230 | 2577.0 | €120.3M |
| Alexis Mac Allister | ARG | Liverpool | MF | 26-136 | 2575.0 | €106.1M |
| Alisson | BRA | Liverpool | GK | 32-219 | 2238.0 | €24.1M |
| Alphonse Areola | FRA | West Ham | GK | 32-071 | 2080.0 | €11.8M |
| Amad Diallo | CIV | Manchester Utd | FW,MF | 22-302 | 1639.0 | €48.5M |
| Amadou Onana | BEL | Aston Villa | MF | 23-266 | 1378.0 | €62.1M |
| Andreas Pereira | BRA | Fulham | MF | 29-128 | 1879.0 | €19M |
| Andrew Robertson | SCO | Liverpool | DF | 31-059 | 2308.0 | €24M |

**Part 4.2: Estimating Player Transfer Values Using Linear Regression.**

- The code selects relevant features for predicting player values, addressing multicollinearity.
- It trains a Linear Regression model.
- It evaluates the model's performance, providing MAE, MSE, and R$^2$ scores.
- Example :



| | Age | Ast | xG | PrgC | PrgP | PrgR | stats_passing_PPA | stats_gca_SCA | stats_gca_SCA90 | stats_gca_GCA | stats_possession_Att Pen | stats_possession_Att | stats_possession_PrgDist | stats_possession_1/3 | stats_possession_CPA | stats_possession_Mis | stats_possession_Dis | stats_misc_Fld | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Age | 1.00 | -0.05 | -0.05 | -0.27 | -0.12 | -0.17 | -0.10 | -0.15 | -0.18 | -0.12 | -0.14 | -0.29 | -0.15 | -0.24 | -0.21 | -0.24 | -0.25 | -0.22 | -0.48 |
| Ast | -0.05 | 1.00 | 0.54 | 0.65 | 0.43 | 0.73 | 0.77 | 0.79 | 0.71 | 0.88 | 0.68 | 0.62 | 0.27 | 0.57 | 0.64 | 0.58 | 0.59 | 0.44 | 0.37 |
| xG | -0.05 | 0.54 | 1.00 | 0.45 | 0.12 | 0.63 | 0.41 | 0.56 | 0.46 | 0.59 | 0.85 | 0.49 | 0.02 | 0.35 | 0.61 | 0.70 | 0.59 | 0.40 | 0.44 |
| PrgC | -0.27 | 0.65 | 0.45 | 1.00 | 0.43 | 0.88 | 0.77 | 0.76 | 0.69 | 0.70 | 0.74 | 0.87 | 0.53 | 0.88 | 0.87 | 0.67 | 0.69 | 0.50 | 0.39 |
| PrgP | -0.12 | 0.43 | 0.12 | 0.43 | 1.00 | 0.25 | 0.65 | 0.67 | 0.47 | 0.52 | 0.21 | 0.30 | 0.63 | 0.65 | 0.12 | 0.25 | 0.34 | 0.48 | 0.40 |
| PrgR | -0.17 | 0.73 | 0.63 | 0.88 | 0.25 | 1.00 | 0.75 | 0.75 | 0.71 | 0.74 | 0.87 | 0.80 | 0.26 | 0.70 | 0.89 | 0.74 | 0.70 | 0.44 | 0.37 |
| stats_passing_PPA | -0.10 | 0.77 | 0.41 | 0.77 | 0.65 | 0.75 | 1.00 | 0.88 | 0.77 | 0.79 | 0.61 | 0.69 | 0.43 | 0.77 | 0.63 | 0.56 | 0.63 | 0.53 | 0.41 |
| stats_gca_SCA | -0.15 | 0.79 | 0.56 | 0.76 | 0.67 | 0.75 | 0.88 | 1.00 | 0.88 | 0.86 | 0.69 | 0.73 | 0.40 | 0.79 | 0.63 | 0.67 | 0.72 | 0.65 | 0.47 |
| stats_gca_SCA90 | -0.18 | 0.71 | 0.46 | 0.69 | 0.47 | 0.71 | 0.77 | 0.88 | 1.00 | 0.78 | 0.62 | 0.67 | 0.18 | 0.68 | 0.62 | 0.57 | 0.65 | 0.51 | 0.41 |
| stats_gca_GCA | -0.12 | 0.88 | 0.59 | 0.70 | 0.52 | 0.74 | 0.79 | 0.86 | 0.78 | 1.00 | 0.73 | 0.67 | 0.34 | 0.66 | 0.67 | 0.63 | 0.66 | 0.56 | 0.46 |
| stats_possession_Att Pen | -0.14 | 0.68 | 0.85 | 0.74 | 0.21 | 0.87 | 0.61 | 0.69 | 0.62 | 0.73 | 1.00 | 0.71 | 0.20 | 0.57 | 0.86 | 0.78 | 0.72 | 0.46 | 0.46 |
| stats_possession_Att | -0.29 | 0.62 | 0.49 | 0.87 | 0.30 | 0.80 | 0.69 | 0.73 | 0.67 | 0.67 | 0.71 | 1.00 | 0.34 | 0.76 | 0.80 | 0.76 | 0.83 | 0.62 | 0.36 |
| stats_possession_PrgDist | -0.15 | 0.27 | 0.02 | 0.53 | 0.63 | 0.26 | 0.43 | 0.40 | 0.18 | 0.34 | 0.20 | 0.34 | 1.00 | 0.60 | 0.30 | 0.12 | 0.16 | 0.24 | 0.33 |
| stats_possession_1/3 | -0.24 | 0.57 | 0.35 | 0.88 | 0.65 | 0.70 | 0.77 | 0.79 | 0.68 | 0.66 | 0.57 | 0.76 | 0.60 | 1.00 | 0.62 | 0.60 | 0.65 | 0.56 | 0.42 |
| stats_possession_CPA | -0.21 | 0.64 | 0.61 | 0.87 | 0.12 | 0.89 | 0.63 | 0.63 | 0.62 | 0.67 | 0.86 | 0.80 | 0.30 | 0.62 | 1.00 | 0.66 | 0.64 | 0.38 | 0.35 |
| stats_possession_Mis | -0.24 | 0.58 | 0.70 | 0.67 | 0.25 | 0.74 | 0.56 | 0.67 | 0.57 | 0.63 | 0.78 | 0.76 | 0.12 | 0.60 | 0.66 | 1.00 | 0.86 | 0.63 | 0.32 |
| stats_possession_Dis | -0.25 | 0.59 | 0.59 | 0.69 | 0.34 | 0.70 | 0.63 | 0.72 | 0.65 | 0.66 | 0.72 | 0.83 | 0.16 | 0.65 | 0.64 | 0.86 | 1.00 | 0.69 | 0.38 |
| stats_misc_Fld | -0.22 | 0.44 | 0.40 | 0.50 | 0.48 | 0.44 | 0.53 | 0.65 | 0.51 | 0.56 | 0.46 | 0.62 | 0.24 | 0.56 | 0.38 | 0.63 | 0.69 | 1.00 | 0.31 |
| Value | -0.48 | 0.37 | 0.44 | 0.39 | 0.40 | 0.37 | 0.41 | 0.47 | 0.41 | 0.46 | 0.46 | 0.36 | 0.33 | 0.42 | 0.35 | 0.32 | 0.38 | 0.31 | 1.00 |



```
Mean Absolute Error (MAE): 11.904445815556226
Mean Squared Error (MSE): 227.40318403347462
R² Score: 0.6553717047107084
```

# 5 CONCLUSION

This project provided a valuable opportunity to explore the intersection of data science and sports analytics through a structured analysis of English Premier League players during the 2024–2025 season. By following a multi-step process - from data collection to predictive modeling - I was able to apply a broad range of Python-based tools and techniques to extract insights and generate meaningful outcomes.

In Part I, player statistics were gathered using web scraping methods and compiled into a unified dataset. Part II involved statistical analysis, where key performance indicators were computed, visualized, and interpreted to highlight individual and team performances. Part III applied machine learning clustering techniques to classify players into distinct groups based on performance metrics, with dimensionality reduction providing a clear visual representation of player clusters.

Finally, in Part IV, I integrated external market data to build a linear regression model that estimates player market values based on selected features, achieving a moderate predictive performance.

This assignment not only reinforced my technical skills in data processing, visualization, and modeling, but also enhanced my ability to think critically about real-world data challenges. While the valuation model demonstrates potential, it also highlights the limitations of relying solely on performance statistics to capture market value - underscoring the influence of external, qualitative factors in player transfers.

In conclusion, this project successfully combined theoretical knowledge with practical implementation, and has deepened my understanding of both Python programming and data science applications in the context of professional sports.

I would like to thank my teacher for their guidance throughout this assignment, as well as my friends and classmates for their support and valuable discussions. Their encouragement helped me stay motivated and complete this project successfully.

Thank you very much!