Ministry of Science and Technology
Posts and Telecommunications Institute of Technology

Assignment Report

Python Programming

Lecturer: Kim Ngoc Bach

Name: Trần Hoàng Nam
Student ID: B23DCCE070
Class: D23CQCE04-B
Group: 04
Tel: 0969486256
Email: NamTH.B23CE070@stu.ptit.edu.vn

Hanoi - 5/2025

Table of contents:

I) Problem 1:

a) Problem analysis:

- Write a Python program to collect footballer player statistical data in 2024-2025 English Premier League season.
  • Data source: https://fbref.com/en/

- Requirements:

• Collect player data who played more than 90 minutes with some statistics about Name, Team, Position, Age, Playing time, Performance, Expected, Progression, Goalkeeping, Shooting, Passing, Goal and Shot Creation, Defensive Actions, Possession and Miscellaneous Stats.

- Save result to file results.csv with requirements:
  o Each column corresponds to a statistic.
  o Players are sorted alphabetically by their first name.
  o Any statistic that is unavailable or inapplicable should be marked as "N/a"

b) Several tools:

- Use selenium to analyze and get data from web
- Use webdriver manager to manage drivers for Selenium
- Use pandas to create Dataframe from players statistics and save to csv
- Use time to wait for web loading

c) Code and Explanation:

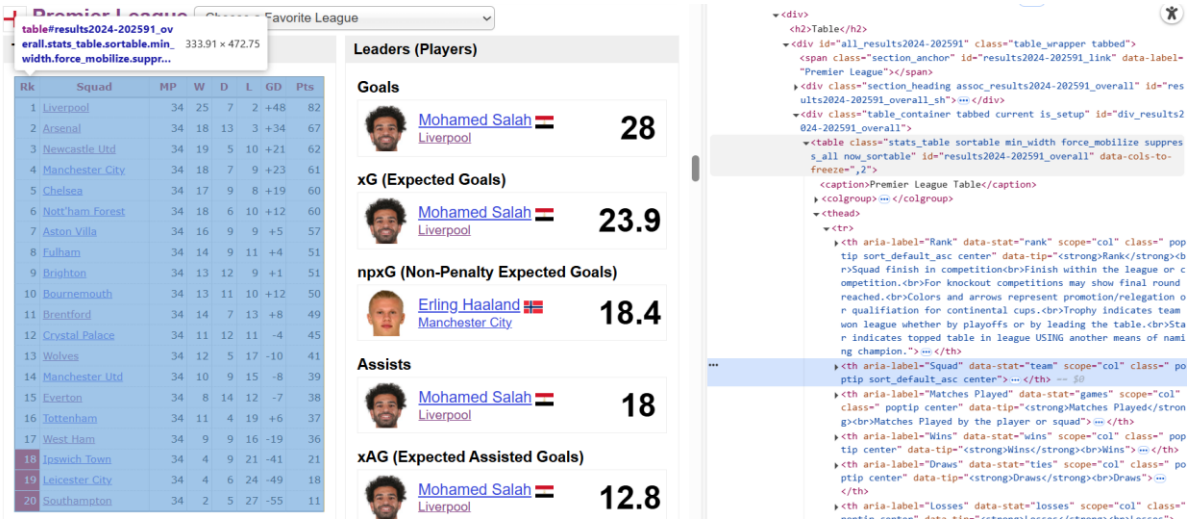
- Firstly, I import libraries and set up a driver:

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager
import pandas as pd

def setup_driver():
    try:
        chrome_options = Options()
        service = Service(ChromeDriverManager().install())
        driver = webdriver.Chrome(service=service, options=chrome_options)
        return driver
    except Exception as e:
        print(f"Failed to setup driver: {e}")
        return None
```

- After that, I get squads link from the source link: https://fbref.com/en/ by using:

```
def get_team_links(driver):
    try:
        print("Getting team links...")
        driver.get("https://fbref.com/en/")
        team_links=[]
        table=driver.find_element(By.ID,'results2024-202591_overall')
        teams=table.find_elements(By.TAG_NAME,'a')
        for team in teams:
            team_links.append(team.get_attribute('href'))
        return team_links

    except Exception as e:
        print(f"Error getting team links: {e}")
        return []
```

- After analysing web by driver, I used find_element(s) to find the table having squad's links. Then, I find all rows having team links by Tag_name 'a' to get 'href' attribute and put it to arr team_links.



- I used get_stat function to get statistics shortly instead of using find_element every time getting stats, that will return text if text exists else ‘N/a’

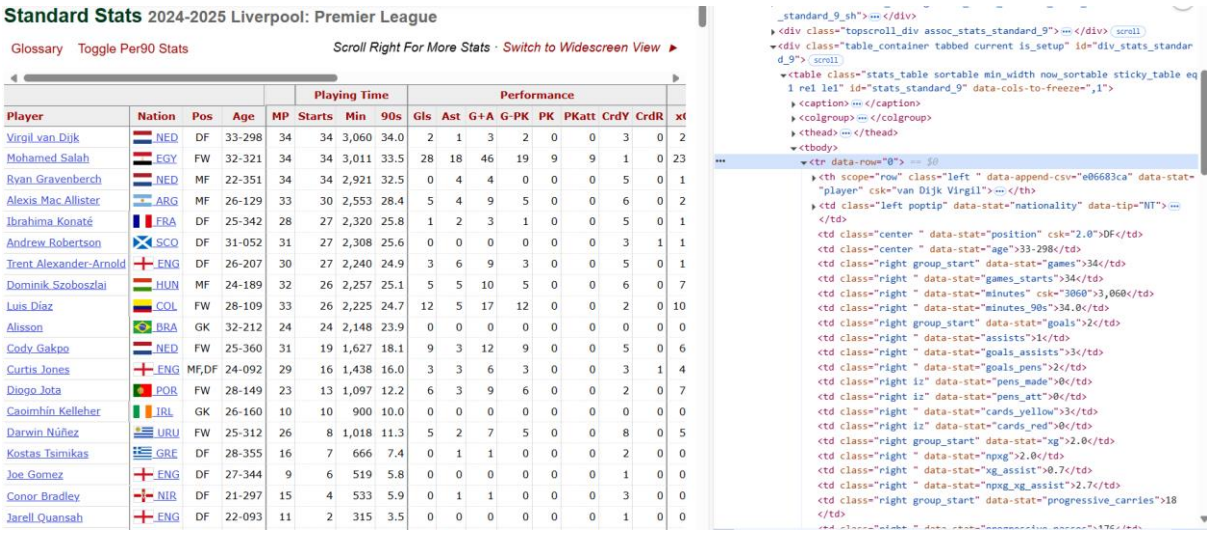
```
def get_stat(row, stat_name):

    try:
        element = row.find_element(By.CSS_SELECTOR, f"td[data-stat='{stat_name}']")
        text = element.text.strip()
        # Return default if text is empty
        if not text:
            return 'N/a'
        return text
    except:
        return 'N/a'
```

- After having squad's links, I will go through each link to get player data in teams.

```
def scrape_team_data(driver, team_url):
    try:
        team_name = team_url.split('/')[-1].replace('-Stats', '')
        print(f"\nScraping team: {team_name}:{team_url}")

        driver.get(team_url)
        time.sleep(2)

        try:
            WebDriverWait(driver, 15).until(
                EC.presence_of_element_located((By.ID, "stats_standard_9"))
            )
        except:
            print(f"Standard stats table not found for {team_name}")
            return []

        players = []
        # Dictionary to store all tables
        tables = {
            'standard': driver.find_element(By.ID, "stats_standard_9") if driver.find_elements(By.ID, "stats_standard_9") else None,
            'keeper': driver.find_element(By.ID, "stats_keeper_9") if driver.find_elements(By.ID, "stats_keeper_9") else None,
            'shooting': driver.find_element(By.ID, "stats_shooting_9") if driver.find_elements(By.ID, "stats_shooting_9") else None,
            'passing': driver.find_element(By.ID, "stats_passing_9") if driver.find_elements(By.ID, "stats_passing_9") else None,
            'gca': driver.find_element(By.ID, "stats_gca_9") if driver.find_elements(By.ID, "stats_gca_9") else None,
            'defense': driver.find_element(By.ID, "stats_defense_9") if driver.find_elements(By.ID, "stats_defense_9") else None,
            'possession': driver.find_element(By.ID, "stats_possession_9") if driver.find_elements(By.ID, "stats_possession_9") else None,
            'misc': driver.find_element(By.ID, "stats_misc_9") if driver.find_elements(By.ID, "stats_misc_9") else None
        }
```

- In this function, I created a arr player that will save a series dictionary containing each player's data.

- Firstly, I created a table thats a dict including data tables of each team.
- To get player's basic data, I will go through all rows in standard table(By.CSS_SELECTOR, "tbody tr:not(.thead)")



Standard Stats 2024-2025 Liverpool: Premier League

Glossary Toggle Per90 Stats Scroll Right For More Stats · Switch to Widescreen View ▸

				Playing Time				Performance								
Player	Nation	Pos	Age	MP	Starts	Min	90s	Gls	Ast	G+A	G-PK	PK	PKatt	CrdY	CrdR	xG
Virgil van Dijk	NED	DF	33-298	34	34	3,060	34.0	2	1	3	2	0	0	3	0	2
Mohamed Salah	EGY	FW	32-321	34	34	3,011	33.5	28	18	46	19	9	9	1	0	23
Ryan Gravenberch	NED	MF	22-351	34	34	2,921	32.5	0	4	4	0	0	0	5	0	1
Alexis Mac Allister	ARG	MF	26-129	33	30	2,553	28.4	5	4	9	5	0	0	6	0	2
Ibrahima Konaté	FRA	DF	25-342	28	27	2,320	25.8	1	2	3	1	0	0	5	0	1
Andrew Robertson	SCO	DF	31-052	31	27	2,308	25.6	0	0	0	0	0	0	3	1	1
Trent Alexander-Arnold	ENG	DF	26-207	30	27	2,240	24.9	3	6	9	3	0	0	5	0	1
Dominik Szoboszlai	HUN	MF	24-189	32	26	2,257	25.1	5	5	10	5	0	0	6	0	7
Luis Díaz	COL	FW	28-109	33	26	2,225	24.7	12	5	17	12	0	0	2	0	10
Alisson	BRA	GK	32-212	24	24	2,148	23.9	0	0	0	0	0	0	0	0	0
Cody Gakpo	NED	FW	25-360	31	19	1,627	18.1	9	3	12	9	0	0	5	0	6
Curtis Jones	ENG	MF,DF	24-092	29	16	1,438	16.0	3	3	6	3	0	0	3	1	4
Diogo Jota	POR	FW	28-149	23	13	1,097	12.2	6	3	9	6	0	0	2	0	7
Caoimhin Kelleher	IRL	GK	26-160	10	10	900	10.0	0	0	0	0	0	0	0	0	0
Darwin Núñez	URU	FW	25-312	26	8	1,018	11.3	5	2	7	5	0	0	8	0	5
Kostas Tsimikas	GRE	DF	28-355	16	7	666	7.4	0	1	1	0	0	0	2	0	0
Joe Gomez	ENG	DF	27-344	9	6	519	5.8	0	0	0	0	0	0	1	0	0
Conor Bradley	NIR	DF	21-297	15	4	533	5.9	0	1	1	0	0	0	3	0	0
Jarell Quansah	ENG	DF	22-093	11	2	315	3.5	0	0	0	0	0	0	1	0	0

- Then, I will filter players with playing minutes over 90, so I get their name and save their statistics to player_rows.

```
standard_table = tables['standard']
if not standard_table:
    return []

player_rows = {}
for row in standard_table.find_elements(By.CSS_SELECTOR, "tbody tr:not(.thead)"):
    try:
        if not row.find_elements(By.CSS_SELECTOR, "th[data-stat='player']"):
            continue

        mins_text = get_stat(row, "minutes", "0")
        mins = int(mins_text.replace(",", "")) if mins_text.replace(",", "").isdigit() else 0
        if mins <= 90:
            continue

        player_name = row.find_element(By.CSS_SELECTOR, "th[data-stat='player']").text
        player_rows[player_name] = {
            'Name': player_name,
            'Nation': get_stat(row, "nationality").split()[-1] if get_stat(row, "nationality") != "N/a" else "N/a",
            'Team': team_name,
            'Position': get_stat(row, "position"),
            'Age': get_stat(row, "age"),
            'Matches': get_stat(row, "games"),
            'Starts': get_stat(row, "games_starts"),
            'Minutes': mins_text,
            'Goals': get_stat(row, "goals"),
            'Assists': get_stat(row, "assists"),
            'Yellow Cards': get_stat(row, "cards_yellow"),
            'Red Cards': get_stat(row, "cards_red"),
            'Expected: xG': get_stat(row, "xg"),
            'Expected: xAG': get_stat(row, "xg_assist"),
            'PrgC': get_stat(row, "progressive_carries"),
            'Progression: PrgP': get_stat(row, "progressive_passes"),
            'Progression: PrgR': get_stat(row, "progressive_passes_received"),
```

- Continuously, I will get the rest of statistics in the other tables(goalkeeping, shooting, passing,…) and attach player's data to matched players.

```
for table_name, table in tables.items():
        if table_name == 'standard' or not table:
            continue
        # Build mapping of players in this table
        table_players = {}
        try:
            for row in table.find_elements(By.CSS_SELECTOR, "tbody tr:not(.thead)"):
              try:
                  player_name = row.find_element(By.CSS_SELECTOR, "th[data-stat='player']").text.strip()
                  if player_name in player_rows:
                    if table_name == 'keeper':
                        player_rows[player_name].update({
                            'GA90': get_stat(row, 'gk_goals_against_per90'),
                            'Save%':get_stat(row, 'gk_save_pct'),
                            'CS%': get_stat(row, 'gk_clean_sheets_pct'),
                            'PK Save%': get_stat(row, 'gk_pens_save_pct'),
                        })
                    elif table_name == 'shooting':
                        player_rows[player_name].update({
                            'SoT%': get_stat(row, "shots_on_target_pct"),
                            'SoT/90': get_stat(row, "shots_on_target_per90"),
                            'G/sh': get_stat(row, "goals_per_shot"),
                            'Dist': get_stat(row, "average_shot_distance"),
                        })
                    elif table_name == 'passing':
                        player_rows[player_name].update({
                            'Cmp': get_stat(row, "passes_completed"),
                            'Cmp%': get_stat(row, "passes_pct"),
                            'TotDist': get_stat(row, "passes_total_distance"),
                            'Short Cmp%': get_stat(row, "passes_pct_short"),
                            'Medium Cmp%': get_stat(row, "passes_pct_medium"),
                            'Long Cmp%': get_stat(row, "passes_pct_long"),
                            'KP': get_stat(row, "assisted_shots"),
                            'Passing: 1/3': get_stat(row, "passes_into_final_third"),
                            'PPA': get_stat(row, "passes_into_penalty_area"),
```

- In main() function, I set up driver and get team_links, after that, I use loop for to go through all links and call scrape_team_data to get player's statistics.

```
def main():
    print("Starting Premier League player data collection...")
    driver = setup_driver()

    if not driver:
        print("Failed to initialize WebDriver")
        return

    try:
        team_links = get_team_links(driver)

        if not team_links:
            print("No team links found")
            return

        all_players = []
        for link in team_links:  # Process all teams
            team_players = scrape_team_data(driver, link)
            if team_players:
                all_players.extend(team_players)
            time.sleep(3)

        data_sorted = sorted(all_players, key=lambda x: x['Name'])
        df=pd.DataFrame(data_sorted,columns=[
        'Name', 'Nation', 'Team', 'Position', 'Age', 'Matches', 'Starts', 'Minutes',
        'Goals', 'Assists', 'Yellow Cards', 'Red Cards', 'Expected: xG', 'Expected: xAG', '
        'Gls', 'Ast', 'xG per 90', 'xGA per 90', 'GA90', 'Save%', 'CS%', 'PK Save%',
        'SoT%', 'SoT/90', 'G/sh', 'Dist', 'Cmp', 'Cmp%', 'TotDist',
        'Short Cmp%', 'Medium Cmp%', 'Long Cmp%', 'KP', 'Passing: 1/3', 'PPA', 'CrsPA','Pas
        'SCA', 'SCA90', 'GCA', 'GCA90', 'Tkl', 'TklW', 'Defensive: Att', 'Defensive: Lost',
        'Sh', 'Pass', 'Int', 'Touches', 'Def Pen', 'Def 3rd', 'Mid 3rd', 'Att 3rd',
        'Att Pen', 'Possession: Att', 'Succ%', 'Tkld%', 'Carries', 'ProDist', 'ProgC','Poss
```

- After collecting all players's data, I'll create a DataFrame and sort it by players's Name. Finally, I will fill blanks in dataFrame with 'N/a' and save to file 'results.csv'.

d) Results:

Result will be saved in file' results.csv' :

Bài 1 > results.csv

	Name	Nation	Team	Position	Age	Matches	Starts	Minutes	Goals	Assists	Yellow Cards	Red Cards	Expected: xG	Expected: xAG	PrgC	Progression: PrgP	Progressi
0	Aaron Cresswell	ENG	West-Ham-United	DF	35-135	14	7	589	0	0	2	0	0.1	1.1	4	24	2
1	Aaron Ramsdale	ENG	Southampton	GK	26-350	26	26	2,340	0	0	2	0	0.0	0.0	0	0	0
2	Aaron Wan-Bissaka	ENG	West-Ham-United	DF	27-154	32	31	2,794	2	2	1	0	1.2	2.9	98	125	139
3	Abdoulaye Doucouré	MLI	Everton	MF	32-118	30	29	2,425	3	1	5	1	3.9	2.3	40	78	91
4	Abdukodir Khusanov	UZB	Manchester-City	DF	21-059	6	6	503	0	0	1	0	0.0	0.1	1	25	2
5	Abdul Fatawu Issahaku	GHA	Leicester-City	FW	21-052	11	6	579	0	2	0	0	0.4	1.6	42	17	60
6	Adam Armstrong	ENG	Southampton	FW,MF	28-078	20	15	1,248	2	2	4	0	3.3	1.2	25	21	79
7	Adam Lallana	ENG	Southampton	MF	36-354	14	5	361	0	2	4	0	0.2	0.9	6	24	10
8	Adam Smith	ENG	Bournemouth	DF	34-000	22	17	1,409	0	0	6	0	0.7	0.3	12	40	31
9	Adam Webster	ENG	Brighton-and-Hove-Albion	DF	30-115	11	8	617	0	0	0	0	0.0	0.5	7	40	2
10	Adam Wharton	ENG	Crystal-Palace	MF	20-331	19	15	1,258	0	2	2	0	0.3	3.0	14	105	10
11	Adama Traoré	ESP	Fulham	FW,MF	29-094	32	16	1,568	2	6	2	0	3.8	4.7	87	61	143
12	Albert Grønbæk	DEN	Southampton	FW,MF	23-341	4	2	143	0	0	0	0	0.1	0.0	1	1	3
13	Alejandro Garnacho	ARG	Manchester-United	MF,FW	20-302	33	22	2,056	5	1	2	0	7.0	3.6	134	54	274
14	Alex Iwobi	NGA	Fulham	FW,MF	28-361	34	32	2,721	9	6	1	0	4.5	6.5	132	196	221
15	Alex McCarthy	ENG	Southampton	GK	35-147	5	5	450	0	0	0	0	0.0	0.0	0	0	0
16	Alex Palmer	ENG	Ipswich-Town	GK	28-262	10	10	900	0	0	2	0	0.0	0.0	0	1	0
17	Alex Scott	ENG	Bournemouth	MF	21-251	17	7	683	0	0	2	0	0.7	0.8	14	48	32
18	Alexander Isak	SWE	Newcastle-United	FW	25-220	31	31	2,487	22	6	1	0	19.0	4.0	77	77	196
19	Alexis Mac Allister	ARG	Liverpool	MF	26-126	33	30	2,553	5	4	6	0	2.8	4.4	34	170	78
20	Ali Al Hamadi	IRQ	Ipswich-Town	FW	23-059	11	0	134	0	0	3	0	0.4	0.1	4	3	14
21	Alisson	BRA	Liverpool	GK	32-209	24	24	2,148	0	0	0	0	0.0	0.6	0	0	0
22	Alphonse Areola	FRA	West-Ham-United	GK	32-061	23	22	1,990	0	0	0	0	0.0	0.0	0	0	0
23	Amad Diallo	CIV	Manchester-United	FW,MF	22-292	22	17	1,594	6	6	3	0	4.1	3.9	92	53	159

II) Problem 2:

a) Problem analysis:

- Identify the top 3 players with the highest and lowest scores for each statistic. Save result to a file name 'top_3.txt'
  + Using nlargest and nsmallest to find top 3 players
  • 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 Atttribute 1	Mean of Atttribute 1	Std of Atttribute 1	...	...
0	all					
1	Team 1					
...						
n	Team n					

+ To solve this problem, we use median(), mean() and std() of pandas to find median, mean and standard of all players and each team, then save results to file 'results2.csv'.

- 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?

b) Several tools to solve:
- Pandas
- Matplotlib.pyplot and seaborn (to plot histogram)
- Os(to create file)

c) Code and Explanation:

- Find top 3 with highest and lowest score

```python
import pandas as pd
df = pd.read_csv('results.csv')

def dfclean(df, stats_col):
    df_cleaned = df.copy()
    for stats in stats_col:
        if stats in df_cleaned.columns:
            df_cleaned[stats] = pd.to_numeric(df_cleaned[stats], errors='coerce')
            df_cleaned[stats] = df_cleaned[stats].fillna(0)
    return df_cleaned

df = df.reset_index()

df = dfclean(df, df.columns[7:])
```

- Initially, I will take data from the previous result, them clean data by dfclean function to choose columns having numeric statistics and reset index .(clean df from columns 7 (numeric statistics))

```python
with open('top_3.txt', 'w', encoding='utf-8') as f:
    for stats in df.columns[7:]:
        other_colls=[ col for col in df.columns if col!=stats and col!='Name']
        display_colls=['Name', stats]+other_colls
        # Get top 3 highest and lowest
        top_high = df.nlargest(3, stats)[display_colls]
        top_low = df.nsmallest(3, stats)[display_colls]

        # Combine into one DataFrame
        combined = pd.concat([top_high.assign(Rank='Top'), top_low.assign(Rank='Bottom')])
        col_widths = {
            'Rank': 8,
            'Index': 10,
            'Name': 20,
            stats: 20,
            'Nation': 12,
            'Team': 18,
            'Position': 15
        }

        header_parts = [
            f"{'Rank':<{col_widths['Rank']}}",
            f"{'Index':<{col_widths['Index']}}",
            f"{'Name':<{col_widths['Name']}}",
            f"{stats:<{col_widths[stats]}}",
            f"{'Nation':<{col_widths['Nation']}}",
            f"{'Team':<{col_widths['Team']}}",
            f"{'Position':<{col_widths['Position']}}"
        ]
```

- I open file ‘top_3.txt’ as f with mode ‘w’ to write data. Then, I will go through columns with required statistics. In the loop, I will create a arr other_colls to save statistics not including the current stat and ‘Name’.
- I re-arrange columns to display with ‘Name’ at first and then the current stat and the other stats.
- I used nlargest and nsmallest of pandas to find top 3 with the highest and lowest stats with the columns as display_cols
- After that, I use concat of pandas to merge 2 above datafram and add a column ‘Rank’ Then I set up statistics’s widths to prettify and created header to display data more beautifully.

```
    headers = "".join(header_parts)
    f.write(headers + '\n')
    f.write('-' * len(headers) + '\n')
    print(headers)
    print('-' * len(headers))

    # Write rows
    for index, row in combined.iterrows():
        # Format stat value
        stat_value = f"{row[stats]:.1f}" if isinstance(row[stats], (int, float)) else str(row[stats])

        row_parts = [
            f"{row['Rank']:<{col_widths['Rank']}}",
            f"{row['index']:<{col_widths['Index']}}",
            f"{str(row['Name'])[:col_widths['Name']]:<{col_widths['Name']}}",
            f"{stat_value:<{col_widths[stats]}}",
            f"{str(row['Nation'])[:col_widths['Nation']]:<{col_widths['Nation']}}",
            f"{str(row['Team'])[:col_widths['Team']]:<{col_widths['Team']}}",
            f"{str(row['Position'])[:col_widths['Position']]:<{col_widths['Position']}}"
        ]


        for col in other_colls[5:]:
            if col != stats:
                cell_value = str(row[col])[:col_widths[col]]
                row_parts.append(f"{cell_value:<{col_widths[col]}}")

        res = "".join(row_parts)
        f.write(res + '\n')
        print(res)

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

- Finally, I write header to file and go through all statistics to write to file with matched widths

Result:

```
Rank    Index   Name                Matches             Nation      Team                Position    Age         Starts      Minutes     Goal
-------------------------------------------------------------------------------------------------------------------------------------
Top     14      Alex Iwobi          34.0                NGA         Fulham              FW,MF       28-361      32          0.0         9
Top     56      Bernd Leno          34.0                GER         Fulham              GK          33-056      34          0.0         0
Top     66      Bruno Guimarães     34.0                BRA         Newcastle-United    MF          27-164      34          0.0         4
Bottom  45      Ayden Heaven        2.0                 ENG         Manchester-United   DF          18-219      1           95.0        0
Bottom  59      Billy Gilmour       2.0                 SCO         Brighton-and-Hove-  MF          23-322      1           98.0        0
Bottom  106     Danny Ward          2.0                 WAL         Leicester-City      GK          31-311      1           135.0       0

Rank    Index   Name                Starts              Nation      Team                Position    Age         Matches     Minutes     Goal
-------------------------------------------------------------------------------------------------------------------------------------
Top     56      Bernd Leno          34.0                GER         Fulham              GK          33-056      34          0.0         0
Top     66      Bruno Guimarães     34.0                BRA         Newcastle-United    MF          27-164      34          0.0         4
Top     111     David Raya          34.0                ESP         Arsenal             GK          29-226      34          0.0         0
Bottom  20      Ali Al Hamadi       0.0                 IRQ         Ipswich-Town        FW          23-059      11          134.0       0
Bottom  35      Antony              0.0                 BRA         Manchester-United   DF,MF       25-064      8           141.0       0
Bottom  48      Ben Chilwell        0.0                 ENG         Crystal-Palace      DF          28-129      6           139.0       0

Rank    Index   Name                Minutes             Nation      Team                Position    Age         Matches     Starts      Goal
-------------------------------------------------------------------------------------------------------------------------------------
Top     86      Christian Eriksen   994.0               DEN         Manchester-United   MF          33-074      20          11          0
Top     384     Paul Onuachu        984.0               NGA         Southampton         FW          30-336      23          11          4
Top     420     Sammie Szmodics     979.0               IRL         Ipswich-Town        FW,MF       29-217      19          13          4
Bottom  1       Aaron Ramsdale      0.0                 ENG         Southampton         GK          26-350      26          26          0
Bottom  2       Aaron Wan-Bissaka   0.0                 ENG         West-Ham-United     DF          27-154      32          31          2
Bottom  3       Abdoulaye Doucouré  0.0                 MLI         Everton             MF          32-118      30          29          3

Rank    Index   Name                Goals               Nation      Team                Position    Age         Matches     Starts      Minu
-------------------------------------------------------------------------------------------------------------------------------------
Top     343     Mohamed Salah       28.0                EGY         Liverpool           FW          32-318      34          34          0.0
Top     18      Alexander Isak      22.0                SWE         Newcastle-United    FW          25-220      31          31          0.0
Top     141     Erling Haaland      21.0                NOR         Manchester-City     FW          24-282      28          28          0.0
```

- Find median, mean and std of all players and each team:

```
def convert_age(age):
    try:
        if '-' in age:
            y, d = map(int, age.split('-'))
            return y + d / 365
        return int(age)
    except:
        return np.nan
df['Age'] = df['Age'].apply(convert_age)
def dfclean(df, stats_col):
    df_cleaned = df.copy()
    for stats in stats_col:
        if stats in df_cleaned.columns:
            df_cleaned[stats] = pd.to_numeric(df_cleaned[stats], errors='coerce')
            df_cleaned[stats] = df_cleaned[stats].fillna(0)
    return df_cleaned
# Keep index as column
df = df.reset_index()

df = dfclean(df, df.columns[6:])
team_names=[]
for  name in df['Team']:
    if not name in team_names:
        team_names.append(name)
team_names.sort()
```

- After reading data from results, I need to convert Age to numeric state(remove '-' in age and clean df. Then, I will take the team names and sort them in dictionary order.

```
for stat in df.columns[6:]:

    all_stats[f'Median of {stat}'] = df[stat].median()
    all_stats[f'Mean of {stat}'] = df[stat].mean()
    all_stats[f'Std of {stat}'] = df[stat].std()
results.append(all_stats)

for team in team_names:
    df_team=df[df['Team']==team]
    data={'':f"{team}"}
    for stat in df.columns[6:]:
        data[f'Median of {stat}'] = df_team[stat].median()
        data[f'Mean of {stat}'] = df_team[stat].mean()
        data[f'Std of {stat}'] = df_team[stat].std()
    results.append(data)

m=pd.DataFrame(results,columns=[i for i in all_stats.keys()])
m.to_csv('results2.csv',index=True,encoding='utf-8-sig')
```

- I use median(), mean() and std() of pandas to solve the problem and save it to results.csv

		Median of Age	Mean of Age	Std of Age	Median of Matches	Mean of Matches	Std of Matches	Median of Starts	Mean of Starts
0	all	26.55890410958904	26.89037189967358	4.231364576874025	22.0	20.672097759674134	9.635415937275416	14.0	15.164969450101832
1	Arsenal	26.86164383561644	26.48169364881694	3.8227439047507037	22.5	22.59090909090909	7.926201822100881	16.0	17.0
2	Aston-Villa	27.61780821917808	27.123091976516633	4.035940846227701	20.0	18.857142857142858	9.965549122460303	9.5	13.357142857142858
3	Bournemouth	25.983561643835618	26.454079809410363	3.8398865412066345	25.0	21.608695652173914	9.838417881100703	17.0	16.217391304347824
4	Brentford	24.82191780821918	26.18317025440313	3.9137679412080035	26.0	22.238095238095237	10.839302384862053	21.0	17.285714285714285
5	Brighton-and-Hove-Albion	24.506849315068493	26.02123287671233	5.170691932287595	20.0	18.964285714285715	9.758119668814247	9.0	13.357142857142858
6	Chelsea	24.295890410958904	24.187881981032664	2.3770583204733398	17.5	19.153846153846153	10.880045248774687	11.5	14.384615384615385
7	Crystal-Palace	27.172602739726027	27.157860404435745	3.134913052545248	29.0	23.38095238095238	10.2101723319256	18.0	17.714285714285715
8	Everton	26.516438356164382	27.95703611457036	5.031962455952982	23.5	21.727272727272727	9.166096007424098	14.5	16.954545454545453
9	Fulham	28.81095890410959	28.80448318804483	3.351455998959689	26.0	23.772727272727273	9.211441220061568	17.0	16.954545454545453
10	Ipswich-Town	26.684931506849317	26.885205479452054	3.1551210970204226	18.0	17.666666666666668	8.742813140471172	11.0	12.466666666666667
11	Leicester-City	26.732876712328768	27.200105374077978	4.46222726444891	21.0	19.73076923076923	9.568940139044418	14.5	14.384615384615385
12	Liverpool	26.43013698630137	27.195564253098503	3.6896849186844096	28.0	24.476190476190474	8.902915520317196	19.0	17.80952380952381
13	Manchester-City	26.671232876712327	26.97928767123288	4.913737779471895	22.0	19.24	8.762039336440653	16.0	14.92
14	Manchester-United	25.71232876712329	25.897818366311512	5.00358148299594	20.0	18.77777777777778	10.966148378369237	14.0	13.777777777777779
15	Newcastle-United	27.44931506849315	27.943895175699822	4.724786370857038	27.0	22.565217391304348	9.917047246381179	13.0	16.26086956521739
16	Nottingham-Forest	27.12054794520548	27.2574097135741	3.593866728645355	28.5	23.136363636363637	10.222735886460582	18.0	16.5
17	Southampton	26.958904109589042	26.95758148323099	4.225866786297973	20.0	18.137931034482758	10.056000830815378	13.0	12.862068965517242
18	Tottenham-Hotspur	25.624657534246577	25.64363267376966	4.538458558710816	21.0	18.88888888888889	9.279146733539896	15.0	13.814814814814815
19	West-Ham-United	28.32054794520548	28.60032876712329	5.015320141012429	20.0	20.92	8.281102986116442	14.0	14.96
20	Wolverhampton-Wanderers	26.838356164383562	27.659678379988087	3.9938137697339573	25.0	22.08695652173913	8.680767893757812	15.0	16.17391304347826

- Plot a histogram showing the distribution of each statistic for all players in the league and each team.

```
# Load data
df = pd.read_csv("results.csv")
def convert_age(age):
    try:
        if '-' in age:
            y, d = map(int, age.split('-'))
            return y + d / 365
        return int(age)
    except:
        return np.nan

def dfclean(df):
    """
    Clean DataFrame by handling NaN and ensuring numeric stats columns.
    """
    df_cleaned = df.copy()
    df_cleaned = df_cleaned.replace("N/a", 0)
    return df_cleaned

# Clean DataFrame
df = dfclean(df)
# Convert Age column
df['Age'] = df['Age'].apply(convert_age)
df['Minutes']=df['Minutes'].str.replace(',','')
df['Minutes']=pd.to_numeric(df['Minutes'],errors='coerce')

if not os.path.exists('Statistics_for_all_players'):
    os.makedirs('Statistics_for_all_players')
if not os.path.exists('Statistics_for_each_team'):
    os.makedirs('Statistics_for_each_team')
atk_def=["SoT/90","G/sh","KP","Tkl","Blocks","Int"]
```

- After taking data from results.csv, I have to convert Age, Minnutes of players into matched forms to plot histogram, then I clean df have replace ‘N/a’ to 0 and use library os

to create file to save histogram. I chose 6 stats to plot histogram: ‘Sot/90’, ‘G/sh’, ‘KP’, ‘Tkl’, ‘Blocks’, ‘Int’ and put them into array atk_def.

```
#===== Part 1: SAVE EACH HISTOGRAM FOR ALL PLAYERS =====

for stat in atk_def:
    stat_name=stat.replace("/", "_").replace(":", "_").replace(" ", "_")
    data = pd.to_numeric(df[stat], errors='coerce').dropna()
    unique_values = data.nunique()
    bins = min(30, unique_values) if unique_values > 1 else 10
    sns.histplot(data,kde=True,bins=bins)
    plt.title(f"Histogram of {stat} for all players")
    plt.ylabel('Frequency')
    plt.xlabel(stat)
    plt.grid(True, alpha=0.3)
    print(f"Histogram of {stat_name} for all players is saved")
    plt.savefig(os.path.join('Statistics_for_all_players',f"{stat_name}.png"))
    plt.show()
    plt.close()

#===== PART 2: SAVE 1 COMBINED HISTOGRAM FOR ALL PLAYERS =====

plt.figure(figsize=(15,10))
for i, stat in enumerate(atk_def,1):
    plt.subplot(2,3,i)
    data=pd.to_numeric(df[stat],errors='coerce').dropna()
    sns.histplot(data,bins=25,kde=True)
    plt.title(f"Histogram of {stat} for all players")
    plt.xlabel(stat)
    plt.ylabel("Frequency")

plt.tight_layout()
plt.savefig(os.path.join('Statistics_for_all_players',f"A combined histogram for all stats.png"))
plt.show()
plt.close()
```

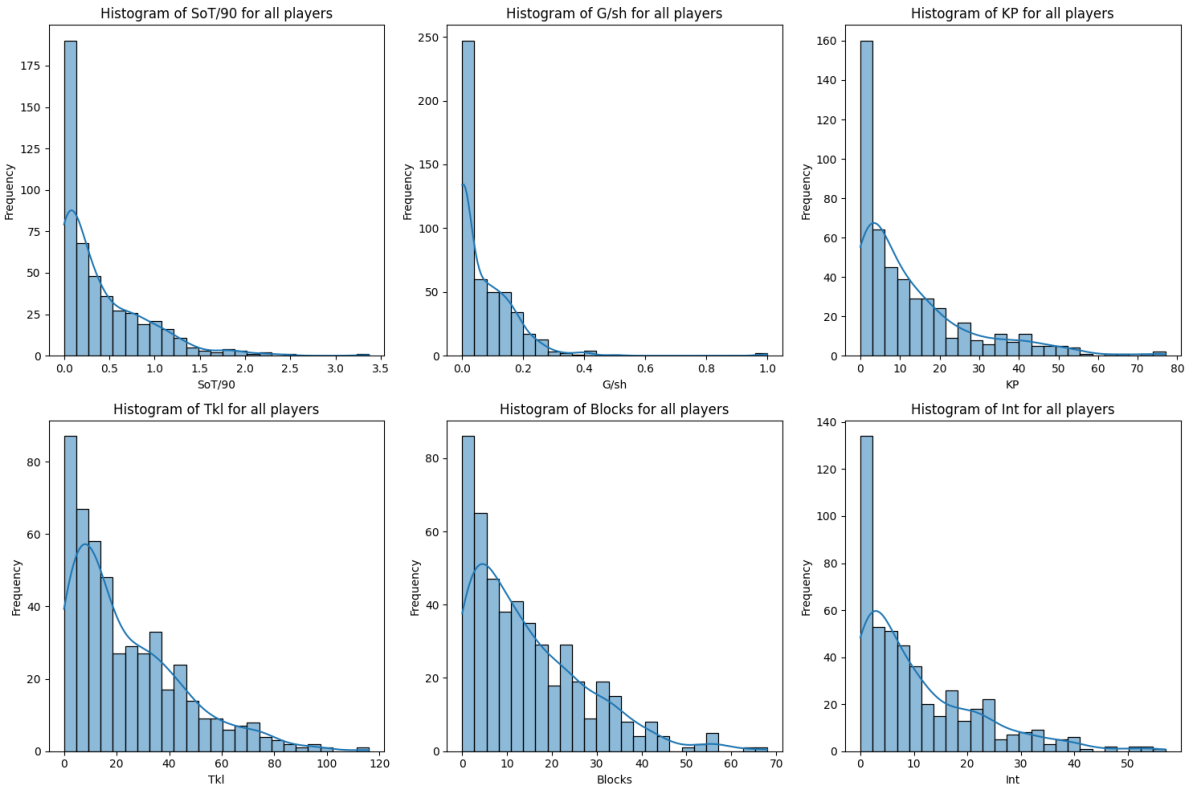
- In this part, I plot histograms for all players and a combined histogram for these above stats for all players. Firstly, I went through each stat and casted the data to numeric before plotting. To choose the numbers of bins, I took min between 30 and the number of unique stats. Then using sns to plot the probability density curve (Kernel Density Estimate - KDE) on the histogram. After labeling X and Y axis, I saved it to Statistics_for_all_players folder

- Same as part 2, I created a larger plt(figsize=(15,10)) to plot each histogram on it. The plt.subplot(2, 3, i) command in matplotlib divides the display area into a grid of cells (subplots), and selects the i-th cell to plot the chart.

- Same as part 3, I plot each combined histogram for each stat

```
df_team=df['Team'].unique()
for team in df_team:
    daf=df[df['Team']==team]
    plt.figure(figsize=(15,10))
    for i,stat in enumerate(atk_def,1):
        stat_name=stat.replace("/", "_").replace(":", "_").replace(" ", "_")
        data=pd.to_numeric(daf[stat],errors='coerce').dropna()
        plt.subplot(2,3,i)
        sns.histplot(data,bins=20,kde=True)
        plt.title(f"{team}")
        plt.xlabel(stat)
        plt.ylabel('Frequency')
    plt.tight_layout()
    plt.savefig(os.path.join('Statistics_for_each_team',f"Histogram for {team}.png"))
    plt.show()
```

- Example a combined histogram for all players:



- 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?

To identify the teams with the highest scores for each stat, I based on the results of results2.csv(saving data of median, mean and std of each team).

```
df = pd.read_csv('results2.csv')
df = df.rename(columns={df.columns[1]: 'Team'})
df = df[df['Team'] != 'all']
print(df)
results = []
team_counts = {}

for stat in df.columns:
    if stat == 'Team':
        continue

    stat_parts = stat.split(' of ')
    stat_type = stat_parts[0] if len(stat_parts) > 1 else 'Value'
    stat_name = stat_parts[1] if len(stat_parts) > 1 else stat

    # Find top performers
    highest_val = df[stat].max()
    highest_teams = df[df[stat] == highest_val]['Team'].tolist()

    results.append({
        'Statistic': stat_name.strip(),
        'Type': stat_type.strip(),
        'Top Teams': ', '.join(highest_teams),
        'Score': highest_val
    })

    # Update team counts (moved outside the loop)
    for team in highest_teams:
        team_counts[team] = team_counts.get(team, 0) + 1
```

- After reading data from csv, I renamed the Team column and delete ‘all’ row. Then, I went through all columns(Median of Age, Mean of Age,…) to deal with data. I splitted stat column into 2 part(eg. Median and Age…) to categorize stat, I found the max value of each stat col and listed the teams having the highest score and put all stat into results then updated the team with the highest score.

```
max_count = max(team_counts.values())
best_teams = [team for team, count in team_counts.items() if count == max_count]

# Generate output
res = pd.DataFrame(results).sort_values(['Statistic', 'Type'])
res.to_csv('Best_teams_stats.csv', index=False)

print("STATISTICAL LEADERS:")
print(res.to_string(index=False))
print("\nTEAM DOMINANCE COUNT:")
print(pd.Series(team_counts).sort_values(ascending=False).to_string())
print(f"\nCONCLUSION: The best-performing team(s) is/are: {', '.join(best_teams)}")
print(f"Leading in {max_count} out of {len(df.columns)-1} statistics")
```

```
Brighton-and-Hove-Albion     7
Manchester-United            7
Ipswich-Town                 7
Brighton-and-Hove-Albion     7
Manchester-United            7
Ipswich-Town                 7

Manchester-United            7
Ipswich-Town                 7

Ipswich-Town                 7

CONCLUSION: The best-performing team(s) is/are: Liverpool

CONCLUSION: The best-performing team(s) is/are: Liverpool
CONCLUSION: The best-performing team(s) is/are: Liverpool
Leading in 81 out of 223 statistics
```

As we can see, there are 223 stats (Ex Age have 3 stat: Median, Mean and std so 74 numeric stat have nearly 223 statistics_

Liverpool lead all with 81 stats per 223

➔ Liverpool is the best team overally

III) Problem 3:

a) Problem analysis:

Use the K-means algorithm to classify players into groups based on thei 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

In this problem, we need to use a machine learning algorithm named Kmeans to classify players.
- Input: A data table containing detailed data about players, including their form, performance and statistics,….

- Output: Groups of players with similar characteristics, with each group representing a type of player with a similar style or role.

- From this, we use PCA to to reduce the data dimensions to 2, then plot a 2D cluster of the data points.

b) Several tools to solve:

- pandas

- matplotlib.pyplot(to plot histogram)

- numpy

- sklearn(to use Kmeans and PCA)

c) Code and Explanation:

- To classify players into groups with similar characteristics, it is necessary to determine the optimal number of groups (clusters) for K-means.

➔ Elbow method: To choose the number of clusters, we can use the Elbow method based on the SSE (Sum of Squared Errors) value. The goal is to choose an inflection point on the graph, where the decrease in SSE starts to level off as the number of clusters increases. This is a sign that adding clusters no longer improves the clustering much.

+ After selecting the number of clusters, we use PCA to reduce the data dimensionality to 2 dimensions for visualization. PCA (Principal Component Analysis) is a popular dimensionality reduction method, which helps retain most of the important information in the original data.

+ Plot a clustering chart: After reducing the dimensionality, we will use K-means to cluster the 2D data. Visualizing the clustering on the 2D plane helps us easily identify groups of players, thereby clearly seeing how the players are classified based on statistical indicators.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from tabulate import tabulate
df = pd.read_csv('results.csv')
dm=df.copy()
def convert_age(age):
    try:
        if '-' in age:
            y, d = map(int, age.split('-'))
            return y + d / 365
        return int(age)
    except:
        return np.nan
df['Age'] = df['Age'].apply(convert_age)
df['Minutes']=df['Minutes'].str.replace(',','')
df['Minutes']=pd.to_numeric(df['Minutes'],errors='coerce')

df = df.iloc[:, 1:]
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
X=df[numeric_cols]
```

- Initially, I import some libraries to use, pandas to create dataframe, matplotlib.pytlot to plot Elbow and chart, sklearn to use KMeans algorithm and PCA to reduce dimension to 2D. I created a copy of df to save the first result from results.csv, after converting Age and Minutes into matched form, I choose the df with numeric columns and fill the blanks with 0.

-

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

imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Step 1: Determine optimal number of clusters using Elbow Method
wcss = []
max_k = 10
for k in range(1, max_k + 1):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

- Then we use StandardScaler to normalize the data so that each feature has: Mean = 0 and Standard deviation = 1.

+ SimpleImputer handles missing values (NaN) in X: with strategy='mean', it replaces each NaN value with the mean of that column. The fit_transform function both calculates the average and applies the replacement. Once NaN has been replaced, it is safe to standardize with StandardScaler.

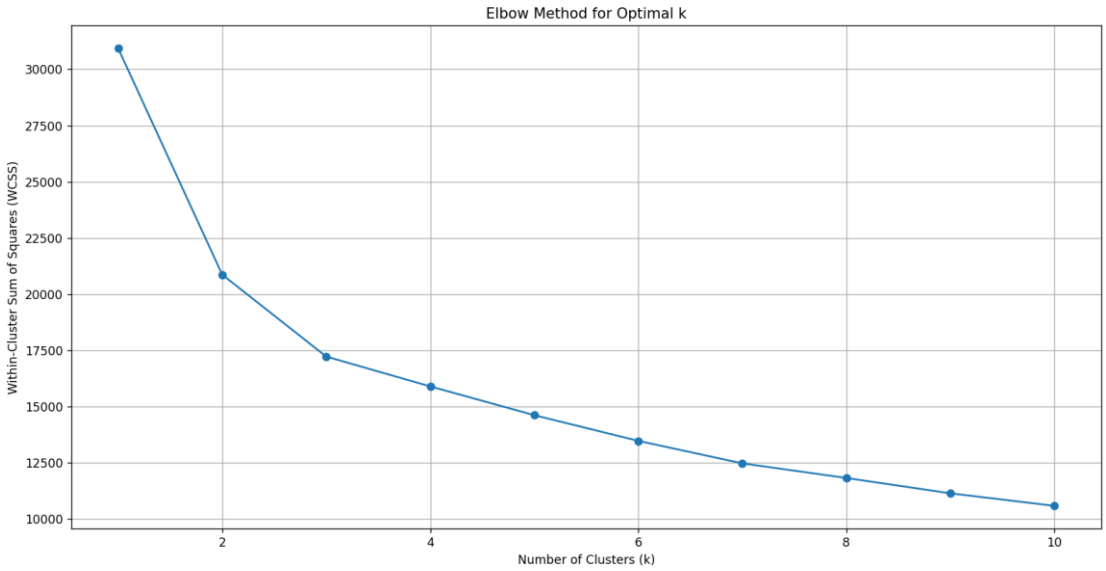
- Step 1, after standardizing, I started determine the number of clusters(K) to classify using Elbow Method.

- I created wcss[] (Within-Cluster Sum of Squares), I attached the first value for k=10 (checking the wcss size from k=1 to k=10).

+ For each cluster number k, calculate a different wcss value: If k increases, the cluster gets smaller → points are closer to the centroid → wcss decreases. For each k, initialize and train the KMeans model. Then get kmeans.inertia_ (which is the wcss for that k) and save it.

```
# Plot Elbow Curve
plt.figure(figsize=(8, 6))
plt.plot(range(1, max_k + 1), wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.grid(True)
plt.savefig('elbow_plot.png')
plt.show()
plt.close()
```

- I used matplotlib to plot histogram of Elbow with the range(1 to 10). So we have:



As we can see, at the point with k=4 to k=5, wcss decreased more slowly than k=1 to k=4.

So k = 4 is a reasonable choice because here:The curve starts to “fold”. After k = 4, WCSS still decreases but the rate of decrease is not significant.

➔ Optimal k=4

Based on k=4, I can choose classify players into 4 roles FW, MF, DF, GK and players playing both role(Ex MF,DF).

```
optimal_k = 4  # Adjust based on elbow plot or domain knowledge
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
cluster_labels = kmeans.fit_predict(X_scaled)

df['Cluster'] = cluster_labels
# Step 3: PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

Create a KMeans object with: n_clusters=4: the number of clusters to find. Random_state=42: to ensure consistent results when running multiple times (replication). Apply the KMeans model to the normalized data (X_scaled). In addition to, I used .fit_predict(...) both trains the model and returns the cluster label for each data row.

Example: [2, 0, 1, 1, 0, 3, ...] (each number is the cluster that the point belongs to).

```
df['Cluster'] = cluster_labels
# Step 3: PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance Ratio by PCA: {explaine explained_variance_ratio_
print(f"Total Variance Explained: {sum(explained_variance):.2%}")
```

- I used PCA to find two main axes (PC1 and PC2) so that when the data is projected onto them, it retains the most of the original variance. This is a way to compress information but still keep the nature of the data.

- Create a PCA object to reduce the data to 2 principal components (PC1 and PC2). n_components=2 → keep only the 2 most important dimensions.

- Next, I use PCA to train on the X_scaled (normalized) data and transform the data into a new 2-dimensional space. The output X_pca is a numpy array of size (n_samples, 2) — each row is the coordinate of a data point (e.g. a player) in the PC1-PC2 space.

Ex: X_pca = [

[5.2, -1.3], # Player 1

[2.1, 0.8], # Player 2

[-3.0, 2.5],

...]

- explained_variance = pca.explained_variance_ratio_

Returns the proportion of variance explained by each principal component.

Example: explained_variance = [0.44147169, 0.20505401]

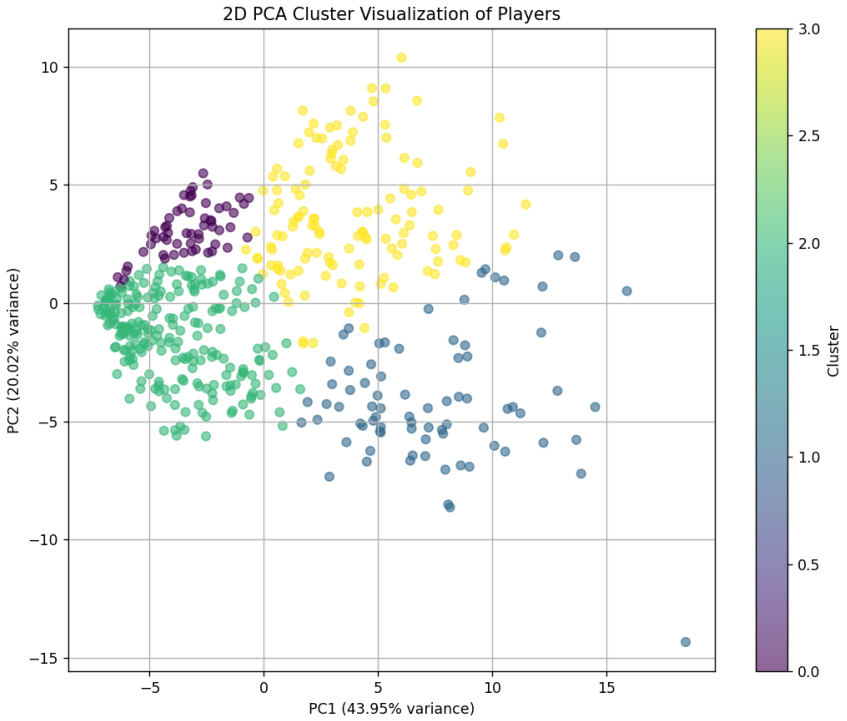
PC1 (first principal axis) explains 44.15% of the original data variance.

PC2 explains another 20.51%.

So, total: 42% + 25% = 67% of information retained.

```
# Plot 2D clusters
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis', alpha=0.6)
plt.colorbar(scatter, label='Cluster')
plt.title('2D PCA Cluster Visualization of Players')
plt.xlabel(f'PC1 ({explained_variance[0]:.2%} variance)')
plt.ylabel(f'PC2 ({explained_variance[1]:.2%} variance)')
plt.grid(True)
plt.savefig('pca_cluster_plot.png')
plt.show()
plt.close()


with open('Clustering_results.txt', 'w', encoding='utf-8-sig') as f:
    for cluster in range(optimal_k):
        print(f"\n========== CLUSTER {cluster} ===========")
        f.write(f"\n========== CLUSTER {cluster} ===========\n\n")
        cluster_df = dm[df['Cluster'] == cluster][dm.columns[1:]] .reset_index(drop=True)
        table_str = tabulate(cluster_df, headers='keys', tablefmt='grid', showindex=True)
        print(cluster_df)
        f.write(table_str)
        f.write("\n\n")


print(f"- Number of clusters chosen: {optimal_k}")
print("- Reasoning: The Elbow Method plot (elbow_plot.png) was analyzed, and k=4 was chosen based on the elbow point and domain knowledge (grouping
print("- PCA Visualization: The 2D plot (pca_cluster_plot.png) shows distinct clusters, though some overlap may exist due to dimensionality reductio
print(f"- Variance Explained: PCA captures {sum(explained_variance):.2%} of the variance, indicating how much information is retained in 2D.")
print("- Cluster Interpretation: Based on sample players, clusters likely correspond to player roles (e.g., high-scoring forwards, defensive players
```

Then, I created a histogram with figsize=(10,8). Use scatter of matplotlib to plot scatter histogram with 2D dimension. The x-axis is the first principal component (X_pca[:, 0]) and the y-axis is the second principal component (X_pca[:, 1]) and save the result to file csv.



Ex: Cluster 0 contains most players playing at Position like: GK, and some DF,MF

========== CLUSTER 0 ============

	Name	Nation	Team	Position	Age	Matches	Starts	Minutes	Goals	Assists
0	Aaron Ramsdale	ENG	Southampton	GK	26-350	26	26	2,340	0	0
1	Alex Palmer	ENG	Ipswich-Town	GK	28-262	10	10	900	0	0
2	Alisson	BRA	Liverpool	GK	32-209	24	24	2,148	0	0
3	Alphonse Areola	FRA	West-Ham-United	GK	32-061	23	22	1,990	0	0
4	André Onana	CMR	Manchester-United	GK	29-027	33	33	2,970	0	0
5	Archie Gray	ENG	Tottenham-Hotspur	DF,MF	19-048	24	15	1,391	0	0
6	Arijanet Muric	KVX	Ipswich-Town	GK	26-173	18	18	1,620	0	0

Cluster 1 contains players playing at Position like : FW,MF

========== CLUSTER 1 ============

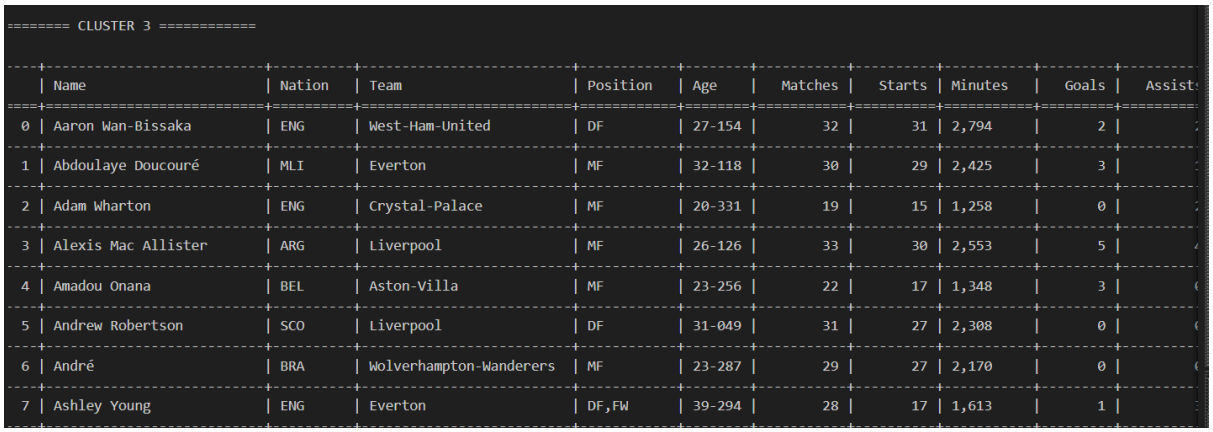
	Name	Nation	Team	Position	Age	Matches	Starts	Minutes	Goals	Assists
0	Adama Traoré	ESP	Fulham	FW,MF	29-094	32	16	1,568	2	6
1	Alejandro Garnacho	ARG	Manchester-United	MF,FW	20-302	33	22	2,056	5	1
2	Alex Iwobi	NGA	Fulham	FW,MF	28-361	34	32	2,721	9	6
3	Alexander Isak	SWE	Newcastle-United	FW	25-220	31	31	2,487	22	6
4	Amad Diallo	CIV	Manchester-United	FW,MF	22-292	22	17	1,594	6	6
5	Andreas Pereira	BRA	Fulham	MF	29-118	31	23	1,879	2	4
6	Anthony Elanga	SWE	Nottingham-Forest	FW	23-002	33	26	2,052	6	9

Cluster 2 contains most players playing at Position like: DF and some FW, MF

========= CLUSTER 2 ===========

	Name	Nation	Team	Position	Age	Matches	Starts	Minutes	Goals	Ass
0	Aaron Cresswell	ENG	West-Ham-United	DF	35-135	14	7	589	0	
1	Abdukodir Khusanov	UZB	Manchester-City	DF	21-059	6	6	503	0	
2	Abdul Fatawu Issahaku	GHA	Leicester-City	FW	21-052	11	6	579	0	
3	Adam Armstrong	ENG	Southampton	FW,MF	28-078	20	15	1,248	2	
4	Adam Lallana	ENG	Southampton	MF	36-354	14	5	361	0	
5	Adam Smith	ENG	Bournemouth	DF	34-000	22	17	1,409	0	
6	Adam Webster	ENG	Brighton-and-Hove-Albion	DF	30-115	11	8	617	0	

Cluster 3 contains most players playing at Position like: MF and some DF



======= CLUSTER 3 ===========

	Name	Nation	Team	Position	Age	Matches	Starts	Minutes	Goals	Assist
0	Aaron Wan-Bissaka	ENG	West-Ham-United	DF	27-154	32	31	2,794	2	
1	Abdoulaye Doucouré	MLI	Everton	MF	32-118	30	29	2,425	3	
2	Adam Wharton	ENG	Crystal-Palace	MF	20-331	19	15	1,258	0	
3	Alexis Mac Allister	ARG	Liverpool	MF	26-126	33	30	2,553	5	
4	Amadou Onana	BEL	Aston-Villa	MF	23-256	22	17	1,348	3	
5	Andrew Robertson	SCO	Liverpool	DF	31-049	31	27	2,308	0	
6	André	BRA	Wolverhampton-Wanderers	MF	23-287	29	27	2,170	0	
7	Ashley Young	ENG	Everton	DF,FW	39-294	28	17	1,613	1	

IV) Problem 4:

a) Problem analysis:
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
In this problem, we can collect player transfer value by finding Premier league link and from this, I will find the most valued table having the list of players who played more than 900 minutes.
In term of missing players(who are not in most valued table), I will find them directly on the search box of web and get transfer value.

b) Several tools to solve:
- Selenium to crawl data
- Pandas to save transfer value to csv file
- Time to sleep while parsing link

c) Code and explanation:

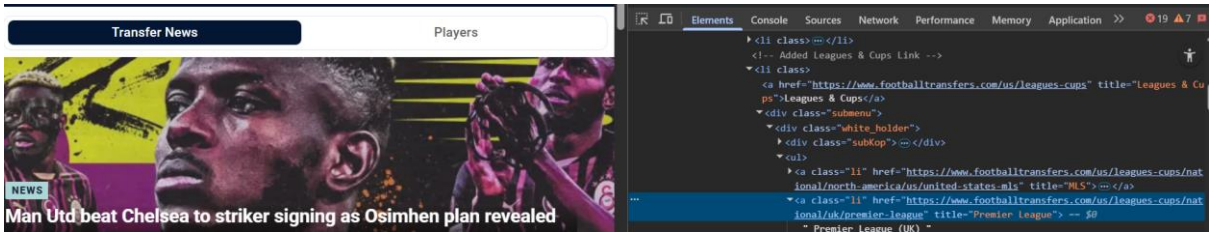
```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service
from bs4 import BeautifulSoup as bs
import pandas as pd
import time
BASE_URL = 'https://www.footballtransfers.com/us'
def load_players():
    try:
        df = pd.read_csv('results.csv')
        df['Minutes'] = df['Minutes'].str.replace(',', '')
        df['Minutes'] = pd.to_numeric(df['Minutes'], errors='coerce')
        df = df[df['Minutes'] > 900].reset_index(drop=True)
        df = df.drop(columns=['Unnamed: 0'], errors='ignore')
        return list(df['Name'])
    except Exception as e:
        print(f"Error loading players: {e}")
        return []

def setup_driver():

    chrome_options = webdriver.ChromeOptions()
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=chrome_options)
    driver.set_page_load_timeout(60)
    return driver
```

- Firstly, we import selenium and webdriver_manager to crawl data. From the previous csv file at the problem 1 – results.csv, I convert Minutes to standard form and get list of names of players who played more than 900 minutes, then set up driver.

-

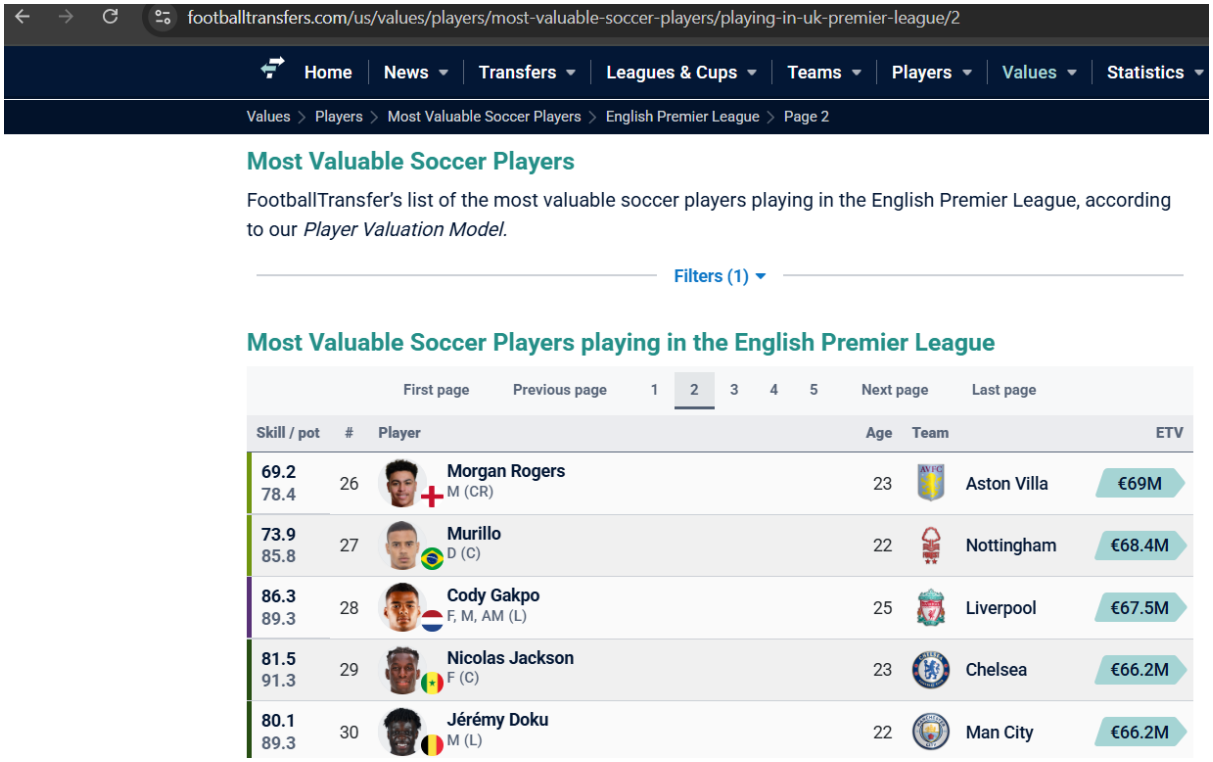
```python
def get_premier_league_link(driver):
    "Get the Premier League players page URL."

    try:
        driver.get(BASE_URL)
        time.sleep(5)
        soup = bs(driver.page_source, 'html.parser')

        pl_tag = soup.find('a', {'title': 'Premier League'})
        if not pl_tag:
            raise Exception("Premier League link not found.")

        return f"{pl_tag.get('href')}/2024-2025"
    except Exception as e:
        print(f"Error getting Premier League link:{e}")

def get_valued_players_link(driver, pl_link):
    "Get the 'all valued players' Premier League URL."
    try:
        driver.get(pl_link)
        time.sleep(3)
        soup = bs(driver.page_source, 'html.parser')

        valued_tag = soup.find('a', {'title': 'View all valued players'})
        if not valued_tag:
            raise Exception("All valued players link not found.")

        return valued_tag.get('href')
    except Exception as e:
        print(f"Error getting valued players link: {e}")
```

- Then, from the initial base link, I got premier league link by using soup in BeautifulSoup to parse page source. After that, I determined a tag with the title ’Premier League’ to get ‘href’ .



- The next step is to access the most valued table having player transfer value after parsing premier link by getting href of tag a with title ‘View all valued players’.

- Because there are 22 pages containing most valuable players, I will process all pages to get value.

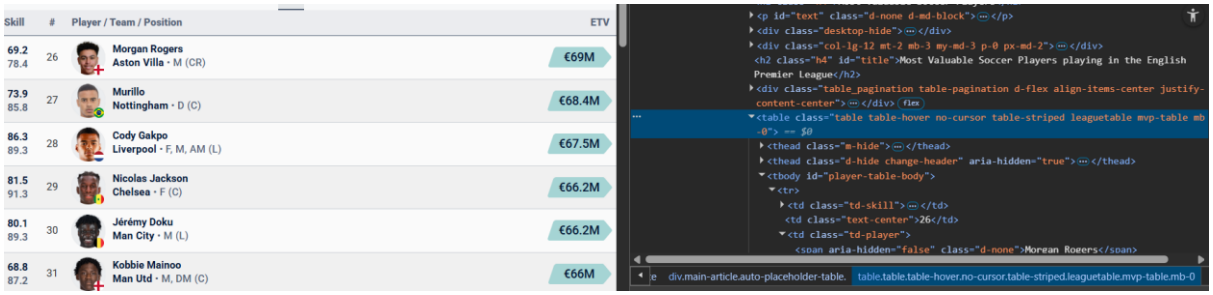



```python
def scrape_player_values(driver, all_players, all_valued_players_link):
    "Scrape player values from the paginated list."
    matched_players = []

    for page in range(1, 23):
            try:
                url = f"{all_valued_players_link}/{page}"
                driver.get(url)
                WebDriverWait(driver, 20).until(
                    EC.presence_of_element_located((By.CSS_SELECTOR, "table.table-hover.no-cursor.table-striped.leaguetable.mvp-table.mb-0"))
                )

                soup = bs(driver.page_source, 'html.parser')
                rows = soup.select('table.table-hover.no-cursor.table-striped.leaguetable.mvp-table.mb-0 tbody#player-table-body tr')

                print(f"Found {len(rows)} rows on page {page}")
                time.sleep(2)
                for row in rows[:1]:
                    print(row)
                    try:
                        td_player = row.find('td', {'class': 'td-player'})
                        if not td_player:
                            continue

                        name_tag = td_player.find('a') or td_player.find('span')
                        if not name_tag:
                            continue
```

- The function scrape_player_values id used to get values of players on each page. All_players is the list of players playing more than 900 mins(will be used to filter player in valued table). All_valued_player_link is the initial link: https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league

- Matched_players is an array to store player name and player value (like {'Name': Alisson, 'Value': '13M'}

- After determining table:



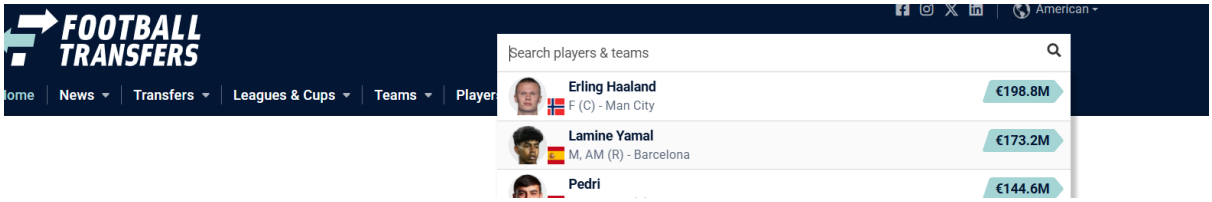
- To find rows of players, I used soup.select('table.table-hover.no-cursor.table-striped.leaguetable.mvp-table.mb-0 tbody#player-table-body tr') to find player name on Tag tbody with all tr.

```
        for row in rows[:1]:
            print(row)
            try:
                td_player = row.find('td', {'class': 'td-player'})
                if not td_player:
                    continue

                name_tag = td_player.find('a') or td_player.find('span')
                if not name_tag:
                    continue

                player_name = name_tag.text.strip()

                if player_name in all_players:
                    value_tag = row.find('span', {'class': 'player-tag'})
                    transfer_value = value_tag.text.strip() if value_tag else "N/A"

                    print(f"Found {player_name}: {transfer_value}")
                    matched_players.append({"Name": player_name, "Value": transfer_value})
                else:
                    print(f" {player_name} not found in list")

            except Exception as e:
                print(f"Error processing row: {e}")

    except Exception as e:
        print(f"Error on page {page}: {e}")

return matched_players
```

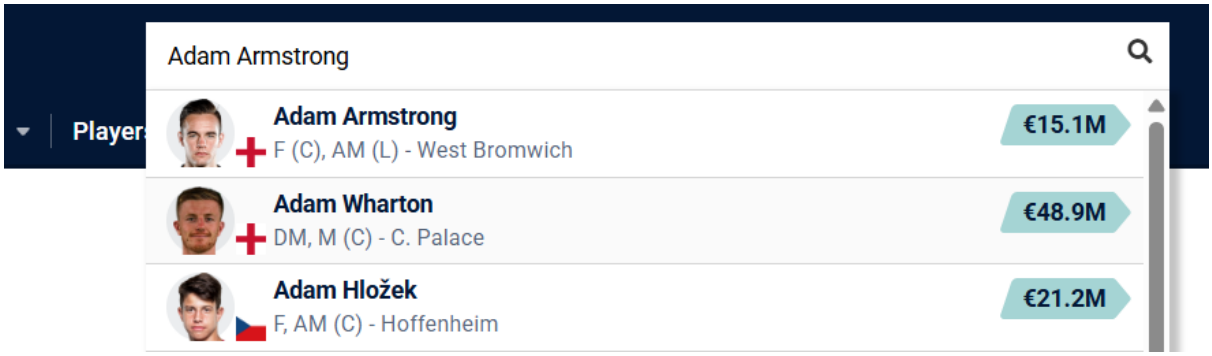
- I processed each row to find player name at td with class='td-player' with Tag 'a' to get text, if this name in all_players(players with more 900 mins), I will take transfer value and return 'N/a' if it didn't exist, and finally return matched_players.

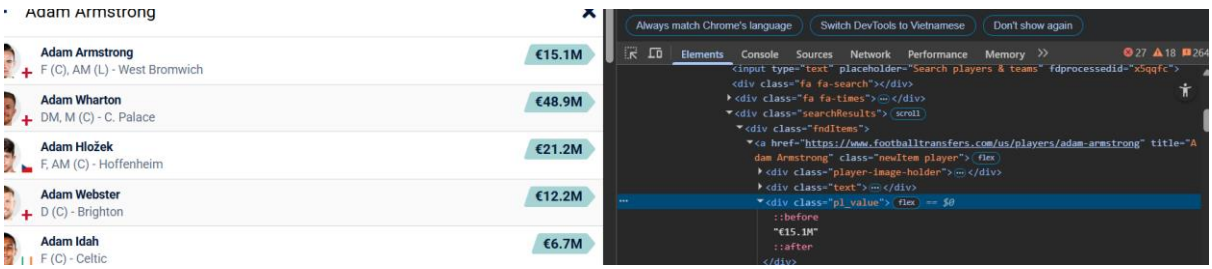
```python
def search_missing_players(driver, player_name):
    """Search for a player not found in the main list."""
    try:
        # Wait for seach box to find and search player name
        search_box = WebDriverWait(driver, 20).until(
            EC.element_to_be_clickable((By.CSS_SELECTOR, "input[type='text'][placeholder*='Search']"))
        )
        search_box.clear()

        for char in player_name:
            search_box.send_keys(char)
            time.sleep(0.2)

        time.sleep(4)
        results = WebDriverWait(driver, 20).until(
            EC.presence_of_all_elements_located((By.CSS_SELECTOR, "div.searchResults a.newItem.player"))
        )

        if results:
            value = results[0].find_element(By.CSS_SELECTOR, "div.pl_value").text
            print(f"Found {player_name} with {value}")
            return value
        else:
            print(f"Not found {player_name} in the result")
            return "Not found"

    except Exception as e:
        print(f"Error when searching {player_name}: {e}")
        return 'Not found'
```

- After finding all players in tables, there were still some missing players not in table. At this step, I would find player value by using search_box.



- After find this box, I will enter name of missing players to get suggestion table in which missing player is on the top and get transfer value





```python
    # Get Premier League valued players page
    pl_link = get_premier_league_link(driver)
    all_valued_players_link = get_valued_players_link(driver, pl_link)
    print(f"Valued players link: {all_valued_players_link}")

    # Scrape player values
    matched_players = scrape_player_values(driver, all_players, all_valued_players_link)

    # Go back to link 'https://www.footballtransfers.com/us' to find values of missing players
    driver.get(BASE_URL)
    time.sleep(3)
# List of missing players
    matched_names = [p['Name'] for p in matched_players]
    missing_players = [p for p in all_players if p not in matched_names]
    print(f"Missing player: {missing_players}")

    for player in missing_players:
        value = search_missing_players(driver, player)
        matched_players.append({"Name": player, "Value": value})

    df = pd.DataFrame(matched_players)
    df['NumericValue'] = df['Value'].apply(normalize_value)
    df = df.sort_values('NumericValue', ascending=False).drop('NumericValue', axis=1)
    df.to_csv('Transfer_value.csv', index=True)
    print(f"\nSaved results to {'Transfer_value.csv'}")

    for _, row in df.iterrows():
        print(f"{row['Name']}: {row['Value']}")
```

- Player name and value will be sorted by transfer value and saved to ‘Transfer_value.csv’ as the result.

	Name	Value
0	Erling Haaland	€199.6M
1	Martin Ødegaard	€125.8M
2	Alexander Isak	€119.4M
3	Cole Palmer	€117.4M
4	Alexis Mac Allister	€117M
5	Declan Rice	€116.4M
6	Bukayo Saka	€113M