POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

REPORT

Course: Programming with Python

Student	Pham Thanh An
Student ID	B23DCCE004
Class	D23CQCE04-B
Course Group	04
Phone Number	0375480845
Email	AnPT.B23CE004@stu.ptit.edu.vn

Hanoi - 05/2025

Index

General introduction

Python is a popular programming language worldwide, known for its clear syntax, readability, and ease of learning. Python supports a wide range of fields, including software development, data analysis, artificial intelligence, and automation. Thanks to its flexibility and high scalability, Python has become a preferred choice for both beginners and professional developers.

The main libraries used in this exercise include:

1. **Requests**: This library helps you send HTTP requests and handle the responses from the web. It's commonly used for web scraping, downloading data from APIs, and interacting with web services.
2. **`bs4` (BeautifulSoup)**: This is a Python library used for parsing HTML and XML. It makes it easy to extract data from web pages. It's often used in combination with Requests for web scraping tasks.
3. **Pandas**: A powerful library for data manipulation and analysis, especially for tabular data (like CSV, Excel, SQL). Pandas provides data structures such as DataFrames, which help in handling, analyzing, and manipulating large datasets efficiently.
4. **Matplotlib (Pyplot)**: Matplotlib is a plotting library for Python. The pyplot module in Matplotlib allows for easy and quick creation of various plots and charts like line, bar, pie, scatter, and more.
5. **Seaborn**: Seaborn is a visualization library based on Matplotlib that provides a more user-friendly interface and aesthetically pleasing charts. It's frequently used for statistical data visualization.
6. **Numpy**: This is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
7. **Scikit-learn**: A powerful library for machine learning in Python. It offers tools for supervised and unsupervised learning algorithms, including classification, regression, clustering, dimensionality reduction, and model evaluation.
8. **Time**: The time library helps with time-related tasks, including measuring time, pausing the program, and converting time formats.
9. **Random**: The random library provides tools for generating random numbers and making random selections from data sets.

All of these libraries are very useful in data analysis, machine learning, and web scraping projects, providing powerful tools for data processing and analysis.

Lesson 1

I. Problem analysis

- Write a Python program to collect footballer player statistical data with the following 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".

II. Required Libraries

- **requests:** Sends HTTP requests to fetch webpage content.
- **bs4 (BeautifulSoup):** Parses HTML to extract data.
- **time:** Pauses the program for a certain amount of time.
- **random:** Generates random values for wait times.
- **pandas:** Saves and exports data to a CSV file.

III. Code

1. Idea

- Visit the Premier League overview page at FBref: https://fbref.com/en/
- Find the leaderboard/results section (<table id="results2024-202591_overall">)
- Iterate through each row (<tr>) to get the link to each team's detail page (the href of the <a> tag is in the data-stat="team" cell).
- Find the corresponding <table id="…">.
- Iterate over the <tr> rows within the <tbody>.
- Get player name (<th data-stat="player">), check if played ≥ 90 minutes (filter through minutes column).
- Each player is stored as a dictionary containing all stats, initializing “N/a” for stats not found.

2. Detailed Description of Key Components

- **Environment Setup and Session Initialization**

```python
session = requests.Session()
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
    'Accept-Language': 'en-US,en;q=0.9'
}
session.headers.update(headers)
```

- Uses the requests library to create a session with headers that mimic a real browser (User-Agent, Accept-Language).
- BeautifulSoup is used to parse the HTML content.

- **Page Loading Function:** get_page(url)

```python
def get_page(url):
    try:
        time.sleep(random.uniform(3, 6))

        response = session.get(url, timeout=60)
        response.raise_for_status()

        if 'text/html' not in response.headers.get('Content-Type', ''):
            raise ValueError("Invalid content type")

        return response.text

    except requests.exceptions.RequestException as e:
        print(f"Request failed: {e}")
        return None
```

- Sends an HTTP request to the given URL.
- Limits request speed using time.sleep(random.uniform(3, 6)) to avoid being blocked.
- Handles connection errors and ensures the returned content is valid HTML.

- **Player Data Structure:** data_player(player_name)

```python
def data_player(player_name):
    return {
        "Name": player_name,
        "Team": "N/a",
        "Nation": "N/A",
        "Position": "N/A",
        "Age": "N/a",
        "Matches Played": "N/a",
        "Starts": "N/a",
        "Minutes": "N/a",
        "Goals": "N/a",
        "Assists": "N/a",
        "Yellow Cards": "N/a",
        "Red Cards": "N/a",
        "Expected Goals (xG)": "N/a",
        "Expected Assist Goals (xAG)": "N/a",
        "Progressive Carries in Progression": "N/a",
        "Progressive Passes in Progression": "N/a",
        "Progressive Passes Received in Progression": "N/a",
        "Goals Scored per 90 minutes": "N/a",
        "Assists per 90 minutes": "N/a",
        "Expected Goals per 90 minutes": "N/a",
        "Expected Assists Goals per 90 minutes": "N/a",
        "Goals Against per 90 minutes": "N/a",
        "Carries": "N/a",
        "Progressive Carry Distance": "N/a",
        "Progressive Carries in Carries": "N/a",
        "Carries into Final Third": "N/a",
        "Carries into Penalty Area": "N/a",
        "Miscontrolls": "N/a",
        "Dispossessed": "N/a",
        "Passes Received": "N/a",
        "Progressive Passes Received in Receiving": "N/a",
        "Fouls Committed": "N/a",
        "Fouls Drawn": "N/a",
        "Offsides": "N/a",
        "Crosses": "N/a",
        "Ball Recoveries": "N/a",
        "Aerial Duels Won": "N/a",
        "Aerial Duels Lost": "N/a",
        "Aerial Duel Win Percentage": "N/a"
    }
```

- Creates a default dictionary storing all information fields for a player, initialized with "N/a" values.

3. **Data Extraction Functions**

```python
> def Standard_Stats(html, team): ...

> def Goalkeeping(html): ...

> def Shooting(html): ...

> def Passing(html): ...

> def Goal_and_Shot_Creation(html): ...

> def Defensive_Actions(html): ...

> def Possession(html): ...

> def Miscellaneous_Stats(html): ...
```

- Each function receives a team's HTML page and extracts a specific statistical table:
+ Standard_Stats: Basic stats (matches played, goals, assists, age, nationality, etc.)
+ Goalkeeping: Goalkeeping stats (save percentage, clean sheets, penalty saves)
+ Shooting: Shooting stats (shots on target rate, goals per match)
+ Passing: Passing stats (total passes, passing accuracy)
+ Goal_and_Shot_Creation: Stats related to shot and goal creation
+ Defensive_Actions: Defensive actions (tackles, blocks)
+ Possession: Ball possession metrics (dribbles, touches)
+ Miscellaneous_Stats: Other stats (offsides, fouls, aerial duels)

These functions use BeautifulSoup to locate the correct HTML table by id and loop through each row (tr) to gather player data.

4. **Team Link Collection**

```
url = "https://fbref.com/en/"
htmls = get_page(url)

if htmls:
    soup = BeautifulSoup(htmls, 'html.parser')
    team_links = {}
    team_count = 1
    total_teams = 20

    if not (table := soup.find('table', {'id': 'results2024-202591_overall'})):
        print("Error: Results table not found")
    else:
        for row in table.find('tbody').find_all('tr'):
            if (team_cell := row.find('td', {'data-stat': 'team'})) and \
               (a_tag := team_cell.find('a')) and \
               a_tag.has_attr('href'):
                team_links[a_tag.text.strip()] = f"{base_url}{a_tag['href']}"
```

- Accesses https://fbref.com/en/.
- Locates the table with id="results2024-202591_overall".
- Loops through each row to collect team names and corresponding links.

5. **Team-by-Team Data Collection and Processing**

```
    for team_name, team_url in team_links.items():
        print(f"Collecting data for {team_name} ({team_count}/{total_teams})...",
              end=' ', flush=True)
        team_count += 1

        if not (team_html := get_page(team_url)):
            print(f"Error: Failed to retrieve {team_url}")
            continue

        try:
            stats_processors = [
                (Standard_Stats, (team_html, team_name)),
                (Goalkeeping, (team_html,)),
                (Shooting, (team_html,)),
                (Passing, (team_html,)),
                (Goal_and_Shot_Creation, (team_html,)),
                (Defensive_Actions, (team_html,)),
                (Possession, (team_html,)),
                (Miscellaneous_Stats, (team_html,))
            ]

            for processor, args in stats_processors:
                processor(*args)
            print("Success!")

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

        time.sleep(random.uniform(5, 10))
```

- For each team:
    + Accesses the team link.
    + Sequentially calls all extraction functions (Standard_Stats, Goalkeeping, etc.).
    + Logs success or errors.
- Introduces random delays of 5–10 seconds after each team to avoid being blocked by the server.

6. **Data Saving**

```
if player_dict:
    output_file = 'results.csv'
    (pd.DataFrame.from_dict(player_dict, orient='index')
       .sort_values(by='Name')
       .to_csv(output_file, index=False, encoding='utf-8-sig'))
    print(f"Success! Data exported to {output_file}")
else:
    print("Warning: No player data was collected")
```

- After completing the loop:
    + Player data is stored in player_dict.
    + Converted into a DataFrame using pandas.
    + Exported to a results.csv file.

IV. Result

- The results are saved in the file 'result.csv'

Name	Team	Nation	Position	Age	Matches Played	Starts	Minutes	Goals	Assists	Yellow Cards	Red Cards	Expected Goals (xG)	Expected Assist Goals (xAG)
Aaron Cresswell	West Ham	ENG	DF	35-133	14	7	589	0	0	2	0	0.1	1.1
Aaron Ramsdale	Southampton	ENG	GK	26-348	26	26	2340	0	0	2	0	0.0	0.0
Aaron Wan-Bissaka	West Ham	ENG	DF	27-152	32	31	2794	2	2	1	0	1.1	2.9
Abdoulaye Doucouré	Everton	MLI	MF	32-116	30	29	2425	3	1	5	1	3.9	2.3
Abdukodir Khusanov	Manchester City	UZB	DF	21-057	6	6	503	0	0	1	0	0.0	0.1
Abdul Fatawu Issahaku	Leicester City	GHA	FW	21-050	11	6	579	0	2	0	0	0.4	1.6
Adam Armstrong	Southampton	ENG	FW,MF	28-076	20	15	1248	2	2	4	0	3.3	1.2
Adam Lallana	Southampton	ENG	MF	36-352	14	5	361	0	2	4	0	0.2	0.9
Adam Smith	Bournemouth	ENG	DF	33-363	21	16	1319	0	0	5	0	0.7	0.2
Adam Webster	Brighton	ENG	DF	30-113	11	8	617	0	0	0	0	0.0	0.5
Adam Wharton	Crystal Palace	ENG	MF	20-329	19	15	1258	0	2	2	0	0.3	3.0
Adama Traoré	Fulham	ESP	FW,MF	29-092	32	16	1568	2	6	2	0	3.8	4.6
Albert Grønbaek	Southampton	DEN	FW,MF	23-339	4	2	143	0	0	0	0	0.1	0.0
Alejandro Garnacho	Manchester Utd	ARG	MF,FW	20-300	32	21	1966	5	1	2	0	6.2	3.5
Alex Iwobi	Fulham	NGA	FW,MF	28-359	34	32	2721	9	6	1	0	4.4	6.5
Alex McCarthy	Southampton	ENG	GK	35-145	5	5	450	0	0	0	0	0.0	0.0
Alex Palmer	Ipswich Town	ENG	GK	28-260	10	10	900	0	0	2	0	0.0	0.0

Lesson 2

I. 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’
- 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					

- 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?

II. Required Libraries

- **pandas:** Read CSV files, manipulate and analyze tabular data (DataFrame)
- **numpy:** Handle missing values (NaN), perform basic numerical operations
- **matplotlib.pyplot**: Plot display and saving
- **seaborn:** Advanced statistical plotting

III. Code

1. Identify the top 3 players with the highest and lowest scores for each statistic.

a. Idea

- Use pandas.read_csv to load results.csv.
- List of columns to consider (stat_cols), convert all to numeric type (to_numeric, error converts to NaN).
- Use nlargest(3) to get the 3 highest values, nsmallest(3) to get the 3

b. Detailed Explanation

- **Data Loading and Cleaning**

```
df = pd.read_csv("results.csv")
df.replace("N/a", np.nan, inplace=True)
```

- Loads the data from the CSV file
- Replaces "N/a" with missing values (NaN) for better numeric processing.

- **Age Conversion**

```
def convert_age(age_str):
    try:
        years, days = map(int, age_str.split("-"))
        return years + days / 365
    except (ValueError, AttributeError):
        return None
```

- The convert_age function converts age strings (like "23-112") into floats.
- A backup column DAge retains the original format.

- **Column Classification**

```
non_stat_cols = ["Name", "Team", "Nation", "Position", "DAge"]
stat_cols = [col for col in df.columns if col not in non_stat_cols]
```

```
df["DAge"] = df["Age"]
df["Age"] = df["Age"].apply(convert_age)
```

- Separates descriptive columns (e.g., name, team) from numerical statistics.

- **Formatting Functions**

```python
def format_header(metric):
    return (
        f"{'Name':<{col_widths['Name']}} "
        f"{'Nation':<{col_widths['Nation']}} "
        f"{'Team':<{col_widths['Team']}} "
        f"{'Position':<{col_widths['Position']}} "
        f"{metric}"
    )

def format_separator(metric):
    return f"{'_'*98} "

def format_row(row, col):
    value = row.get("DAge", "") if col == "Age" else f"{row[col]:.2f}"
    return (
        f"{row['Name']:<{col_widths['Name']}} "
        f"{row['Nation']:<{col_widths['Nation']}} "
        f"{row['Team']:<{col_widths['Team']}} "
        f"{row['Position']:<{col_widths['Position']}} "
        f"{value}"
    )
```

- These functions ensure aligned, clean formatting of each output row for consistent text display.

- **Extract Top 3 and Bottom 3**

```python
for col in stat_cols:
    if col not in df.columns or not pd.api.types.is_numeric_dtype(df[col]):
        continue

    cols_to_use = display_cols + ([col] if col not in display_cols else [])
    if col == "Age" and "DAge" not in cols_to_use:
        cols_to_use.append("DAge")
    temp_df = df[cols_to_use].dropna(subset=[col])

    if temp_df.empty:
        continue

    try:
        top3 = temp_df.nlargest(3, col)
        bottom3 = temp_df.nsmallest(3, col)
    except Exception as e:
        print(f"Error {col}: {e}")
        continue
```

- For each numeric metric, the script extracts the top and bottom 3 players using nlargest and nsmallest .

- **Output to Text File**

```
try:
    with open("top_3.txt", "w", encoding="utf-8") as f:
        f.write("\n".join(output))
    print("Top 3 and bottom 3 statistics saved to 'top_3.txt'")
except Exception as e:
    print(f"Error saving 'top_3.txt': {e}")
```

- Writes the formatted results to a text file for review or presentation.

2. Find the median for each statistic. Calculate the mean and standard deviation for each statistic across all players and for each team.

a. Idea

- Used a comprehension to iterate over each column col in stat_cols and each function func in ['median', 'mean', 'std']
- Collected the results into a dictionary global_metrics with keys like "Median of Goals", "Mean of Assists"
- For each statistic column and each function ('median', 'mean', 'std'), df[col].agg(func) internally calls df[col].median(), df[col].mean(), or df[col].std()

b. Detailed Explanation

- **Data Loading**

```
df = pd.read_csv("results.csv")
```

- Load the CSV data into a pandas DataFrame.

- **Column Separation**

```
non_stat = ["Name", "Nation", "Team", "Position"]
stat_cols = [c for c in df.columns if c not in non_stat]
```

- Non-statistical columns are identified.
- Remaining columns are assumed to contain numeric data for analysis.

- **Global Statistics**

```python
funcs = ['median', 'mean', 'std']

global_metrics = {
    f"{func.capitalize()} of {col}": df[col].agg(func)
    for col in stat_cols
    for func in funcs
}
global_df = pd.DataFrame(global_metrics, index=['All'])
```

- Calculates the mean, median, and standard deviation for all players across each stat column.

- **Per-Team Aggregation**

```python
team_df = df.groupby("Team")[stat_cols].agg(funcs)
team_df.columns = [f"{func.capitalize()} of {col}" for col, func in team_df.columns]
```

- Groups players by "Team".
- Calculates descriptive statistics for each team.

- **Merging and Exporting Results**

```python
result = pd.concat([global_df, team_df], axis=0)

result = result.reset_index()
result.rename(columns={"index": "Team"}, inplace=True)

result.to_csv("results2.csv", index=True, encoding='utf-8-sig')
```

- Merges global and team-level results.
- Renames and resets the index to ensure "Team" is properly labeled.
- Exports the final DataFrame to results2.csv.

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

a. Idea

- Use seaborn.histplot() to draw 6 histograms (3 attacking + 3 defensive) in a figure arranged in 2 rows and 3 columns.
- The kde=True parameter adds a kernel density estimation line to better illustrate the distribution.

b. Detailed Explanation

- **Loading the data**

```
df = pd.read_csv("results.csv")
```

- Reads the dataset from the results.csv file into a Pandas DataFrame.

- **Selecting key metrics**

```
# Chọn đại diện 3 chỉ số tấn công và 3 chỉ số phòng thủ
attack_stats  = ['Goals', 'Assists', 'Expected Goals (xG)']
defense_stats = ['Tackles', 'Interceptions', 'Blocks']
selected_stats = attack_stats + defense_stats
```

- 3 attacking metrics: Goals, Assists, Expected Goals (xG).
- 3 defensive metrics: Tackles, Interceptions, Blocks.

- **Setting plot styles**

```
sns.set_style("whitegrid")
plt.rcParams['font.size'] = 10
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['axes.titleweight'] = 'bold'
```

- Configures the plot aesthetics: light grid background, font sizes, bold axis titles.

- **Plotting for all players**

```
plt.figure(figsize=(14, 8))
plt.suptitle('', fontsize=16, fontweight='bold', x=1.02, y=1.02)
for i, stat in enumerate(selected_stats, 1):
    plt.subplot(2, 3, i)
    sns.histplot(df[stat].dropna(), kde=True, bins=20,
                 color='skyblue', edgecolor='w', linewidth=0.5)
    plt.title(f'{stat}')
    plt.xlabel('')
plt.tight_layout(pad=2.5)
plt.suptitle('All Players', fontsize=30, x=0.5, y=0.98)
plt.savefig('Histograms_all_players.png', bbox_inches='tight')
print(f"Saved to file: {'Histograms_all_players.png'}")
plt.show()
```

- Generates one histogram per selected statistic using all players.
- The result is saved as Histograms_all_players.png.

- **Defining team order**

```
teams_order = [
    "Liverpool", "Arsenal", "Manchester City", "Nott'ham Forest",
    "Newcastle Utd", "Chelsea", "Aston Villa", "Bournemouth",
    "Fulham", "Brighton", "Brentford", "Crystal Palace",
    "Everton", "Manchester Utd", "Wolves", "Tottenham",
    "West Ham", "Ipswich Town", "Leicester City", "Southampton"
]
```

- A manually defined list of team names to control the order of plotting.

- **Plotting by team**

```
for team in teams_order:
    if team not in df['Team'].values:
        continue

    team_df = df[df['Team'] == team]

    plt.figure(figsize=(14, 8))
    plt.suptitle(team, y=1.02, fontsize=14, fontweight='bold')

    for i, stat in enumerate(selected_stats, 1):
        ax = plt.subplot(2, 3, i)
        sns.histplot(team_df[stat].dropna(), kde=True, bins=20,
                     color='skyblue', edgecolor='w', linewidth=0.5)
        plt.title(f'{stat}')
        plt.xlabel('')

    plt.tight_layout(pad=2.5)
    plt.suptitle(f'{team}', fontsize=30, x=0.5, y=0.96)
    filename = f'Histograms_{team.replace(" ", "_").replace("/", "-")}.png'
    plt.savefig(filename, bbox_inches='tight')
    print(f"Saved to file: {filename}")
    plt.show()
```

- Iterates through each team in the teams_order list.
- Filters the DataFrame to only include players from that team.
- Plots the same statistics and saves each output as a separate file.

4. 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?

a. Idea

- For each column that starts with "Mean of", find the team with the highest value.
- Sum all "Mean of ..." columns to create a new column "Total Mean Score" — representing overall team performance.

b. Detailed Explanation

- **Load the dataset**

```
df = pd.read_csv('results2.csv')
```

- Load the file results2.csv into a pandas DataFrame df.

- **Prepare structures for analysis**

```
Team_mean = []

df['Total Mean Score'] = 0
```

- Create an empty list Team_mean to store top teams by attribute.
- Add a new column 'Total Mean Score' initialized to 0.

- **Find the team with the highest average score for each attribute**

```
df['Total Mean Score'] = 0

for column_name in df.columns:
    if column_name.startswith('Mean of'):
        team_with_highest_mean = df.loc[df[column_name].idxmax(), 'Team']

        highest_mean_value = df[column_name].max()

        Team_mean.append({
            'Attribute': column_name.replace('Mean of ', ''),
            'Top Team': team_with_highest_mean,
            'Mean Value': highest_mean_value
        })

        print(f"The team with the highest '{column_name.replace('Mean of ', '')}' is '{team_with_highest_mean}', with Mean Value: {highest_mean_val

        df['Total Mean Score'] += df[column_name].fillna(0)
```

- Create an empty list Team_mean to store top teams by attribute.
- Add a new column 'Total Mean Score' initialized to 0.

- **Identify top team by attribute**

```
team_with_best_total_performance = df.loc[df['Total Mean Score'].idxmax(), 'Team']
highest_total_mean_score = df['Total Mean Score'].max()
```

- Get the team name with the highest mean value in the current column.

- Store the highest value itself.

- **Save result**

```python
df_team = pd.DataFrame(Team_mean)

output_file = 'Top_Teams.csv'

df_team.to_csv(output_file, index=False, encoding='utf-8-sig')

print(f"\n📁 The results have been saved to the file: {output_file}")
```

- Create a DataFrame from the Team_mean list and save it to the file Top_Teams.csv.
- Print out the results for each metric and the best team of the season.

IV. Results

- The results are saved in the file:
  - ‘top_3.txt’

```
══════════════════════════════════ Age ══════════════════════════════════
TOP 3:
Name                          Nation     Team                 Position     Age
──────────────────────────────────────────────────────────────────────────────────────────────
Łukasz Fabiański              POL        West Ham             GK           40-009
Ashley Young                  ENG        Everton              DF,FW        39-292
James Milner                  ENG        Brighton             MF           39-113
----------------------------------------------------------------------------------------------
BOTTOM 3:
Name                          Nation     Team                 Position     Age
──────────────────────────────────────────────────────────────────────────────────────────────
Mikey Moore                   ENG        Tottenham            FW,MF        17-259
Ethan Nwaneri                 ENG        Arsenal              FW,MF        18-037
Harry Amass                   ENG        Manchester Utd       DF           18-042


══════════════════════════════════ Matches Played ══════════════════════════════════
TOP 3:
Name                          Nation     Team                 Position     Matches Played
──────────────────────────────────────────────────────────────────────────────────────────────
Alex Iwobi                    NGA        Fulham               FW,MF        34.00
Bernd Leno                    GER        Fulham               GK           34.00
Bruno Guimarães               BRA        Newcastle Utd        MF           34.00
----------------------------------------------------------------------------------------------
BOTTOM 3:
Name                          Nation     Team                 Position     Matches Played
──────────────────────────────────────────────────────────────────────────────────────────────
Altay Bayındır                TUR        Manchester Utd       GK           1.00
Brandon Austin                ENG        Tottenham            GK           1.00
```

  - ‘results2.csv’

	Team	Median of Age	Mean of Age	Std of Age	Median of Matches Played	Mean of Matches Played	Std of Matches Played	Median of Starts	Mean of Starts
0	All	26.54794520547945	26.87582149760484	4.2470172602087475	22.0	20.443762781186095	9.776047958214853	14.0	15.042944785276074
1	Arsenal	26.856164383561644	26.476214196762143	3.8227439047507046	22.5	22.59090909090909	7.926201822100882	16.0	17.0
2	Aston Villa	27.715068493150685	27.117199391171994	4.112822214879278	20.0	19.444444444444443	9.775689365160012	12.0	14.074074074074074
3	Bournemouth	25.97808219178082	26.44860035735557	3.8398865412066323	24.0	21.0	9.496410805236795	16.0	15.73913043478261
4	Brentford	24.816438356164383	26.177690802348337	3.9137679412080026	26.0	22.238095238095237	10.839302384862052	21.0	17.285714285714285
5	Brighton	24.5013698630137	26.015753424657532	5.170691932287595	20.0	18.678571428571427	10.015001974551204	9.0	13.571428571428571
6	Chelsea	24.29041095890411	24.18240252897787	2.3770583204733398	17.5	19.23076923076923	10.80484221932997	12.5	14.538461538461538
7	Crystal Palace	27.167123287671235	26.964253098499675	3.3317964091213197	29.0	22.857142857142858	11.069262460912716	18.0	17.19047619047619
8	Everton	25.912328767123288	27.620369267421083	5.166475082857304	22.0	20.82608695652174	9.943715118391678	14.0	16.26086956521739
9	Fulham	28.805479452054794	28.799003735990038	3.3514559989596875	25.0	22.681818181818183	10.40323376298316	16.5	15.863636363636363
10	Ipswich Town	26.95205479452055	27.211448140900195	2.984893942993378	18.0	18.214285714285715	8.799651267741377	11.5	12.857142857142858
11	Leicester City	26.72739726027397	27.194625922023185	4.46222726444891	21.0	19.73076923076923	9.568940139044418	14.5	14.384615384615385
12	Liverpool	26.424657534246574	27.190084801043707	3.689684918684409	27.0	23.714285714285715	8.5623761722016	18.0	17.285714285714285
13	Manchester City	26.665753424657535	26.973808219178082	4.913737779471895	22.0	19.24	8.762039336440653	16.0	14.92
14	Manchester Utd	25.693150684931506	25.519330289193302	5.051452934014777	20.0	17.62962962962963	11.398880336878397	14.0	13.0
15	Newcastle Utd	27.443835616438356	27.93841572364503	4.724786370857037	27.0	22.565217391304348	9.917047246381177	13.0	16.26086956521739
16	Nott'ham Forest	27.115068493150684	27.251930261519302	3.593866728645354	28.5	23.227272727272727	10.099183456770477	18.0	16.681818181818183
17	Southampton	26.953424657534246	26.952102031176192	4.225866786297974	20.0	18.137931034482758	10.05600083081538	13.0	12.862068965517242
18	Tottenham	25.958904109589042	25.66174168297456	4.455368831383845	19.0	17.714285714285715	9.552417351051892	14.0	12.964285714285714

- Top_Teams.csv

Attribute	Top Team	Mean Value
Age	West Ham	28.86372840976772
Matches Played	Liverpool	23.714285714285715
Starts	Brentford	17.285714285714285
Minutes	Liverpool	1549.2380952380952
Goals	Liverpool	3.571428571428572
Assists	Liverpool	2.6666666666666665
Yellow Cards	Chelsea	3.6538461538461537
Red Cards	Arsenal	0.2272727272727272
Expected Goals (xG)	Liverpool	3.523809523809524
Expected Assist Goals (xAG)	Liverpool	2.538095238095238
Progressive Carries in Progression	Manchester City	40.56
Progressive Passes in Progression	Liverpool	79.57142857142857
Progressive Passes Received in Progression	Liverpool	78.80952380952381
Goals Scored per 90 minutes	Aston Villa	0.1948148148148148
Assists per 90 minutes	Liverpool	0.1466666666666666
Expected Goals per 90 minutes	Aston Villa	0.1911111111111111
Expected Assists Goals per 90 minutes	Chelsea	0.1507692307692307
Goals Against per 90 minutes	Leicester City	2.73
Save Percentage	Bournemouth	79.86666666666666
Clean Sheets Percentage	Brentford	57.8
Penalty Save Percentage	Everton	100.0
Percentage Of Shots That Are On Target	Nott'ham Forest	38.015

Lesson 3

I. Problem analysis

- 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.

II. Required Libraries

- **pandas:** Reading and manipulating tabular data (CSV)
- **numpy:** Numerical computations (not directly used here but commonly paired with pandas)
- **sklearn.preprocessing.StandardScaler:** Standardizes data for better clustering performance
- **sklearn.cluster.KMeans:** Performs K-Means clustering
- **sklearn.decomposition.PCA:** Reduces dimensions for visualization
- **sklearn.metrics.silhouette_score:** Evaluates clustering quality
- **matplotlib.pyplot:** Data visualization (Elbow curve, PCA scatter)
- **seaborn:** Advanced data visualization (colored scatterplots)

III. Code

1. Introduction

- Use StandardScaler to normalize the data (mean = 0, standard deviation = 1).
- Loop through k from 2 to 9.
- For each k, train a KMeans model:
  - Calculate **Inertia** (sum of squared distances from each point to its cluster center) — used for the **Elbow Method**.
  - Calculate **Silhouette Score** — measures how well-separated and cohesive the clusters are.
- Plot the **Elbow graph** and **Silhouette scores** to identify the best k.
- Use **PCA** to reduce the feature space to 2 dimensions (PCA1, PCA2).
- Create a 2D scatter plot to visualize how the clusters are distributed.

2. Detailed Description of Key Components

- **Load and prepare data**

```
data = pd.read_csv('results.csv')

numeric_features = data.select_dtypes(include=['float64', 'int64'])
```

- Loads the dataset from CSV.
- Selects numeric columns (float and int) for clustering.

- **Standardize the data**

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_features)
```

- Standardizes the data to mean = 0 and standard deviation = 1 using StandardScaler, which improves clustering performance.

- **Determine the optimal number of clusters**

```
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(X_scaled)
    inertia_values.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, cluster_labels))
```

- Tries cluster values from 2 to 10.
- For each value of k, it:
    + Applies K-Means.
    + Calculates:
        - Inertia (within-cluster distance).
        - Silhouette Score (how well clusters are separated).

- **Plot Elbow and Silhouette graphs**

```
fig, ax = plt.subplots(1, 2, figsize=(16, 6))

ax[0].plot(range(2, 11), inertia_values, marker='o', linestyle='--')
ax[0].set_xlabel('Number of clusters (k)')
ax[0].set_ylabel('Inertia')
ax[0].set_title('Elbow Method')
ax[0].grid(True)

ax[1].plot(range(2, 11), silhouette_scores, marker='o', linestyle='--', color='b')
ax[1].set_xlabel('Number of clusters (k)')
ax[1].set_ylabel('Silhouette Score')
ax[1].set_title('Silhouette Score')
ax[1].grid(True)

plt.tight_layout()
plt.savefig('Kmeans.png', bbox_inches='tight')
plt.show()
```

- Plots:
    + Elbow Method to find where inertia starts to level off.
    + Silhouette Score to find the highest score indicating the best cluster separation.

- **Select optimal number of clusters**

```
optimal_k = 4

kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)
```

- Applies K-Means with k=4 (determined manually or from plots).
- Stores cluster labels for each record.

- **Apply PCA for visualization**

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

pca_df = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2'])
pca_df['Cluster'] = cluster_labels

plt.figure(figsize=(10, 7))
sns.scatterplot(data=pca_df, x='PCA1', y='PCA2', hue='Cluster', palette='Set1', s=100)
plt.title('K-Means Clustering Results (after PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.savefig('Kmeans_PCA.png', bbox_inches='tight')
plt.show()
```

- Reduces dimensionality to 2 principal components (PCA1, PCA2) for plotting.
- Creates a scatterplot colored by cluster.

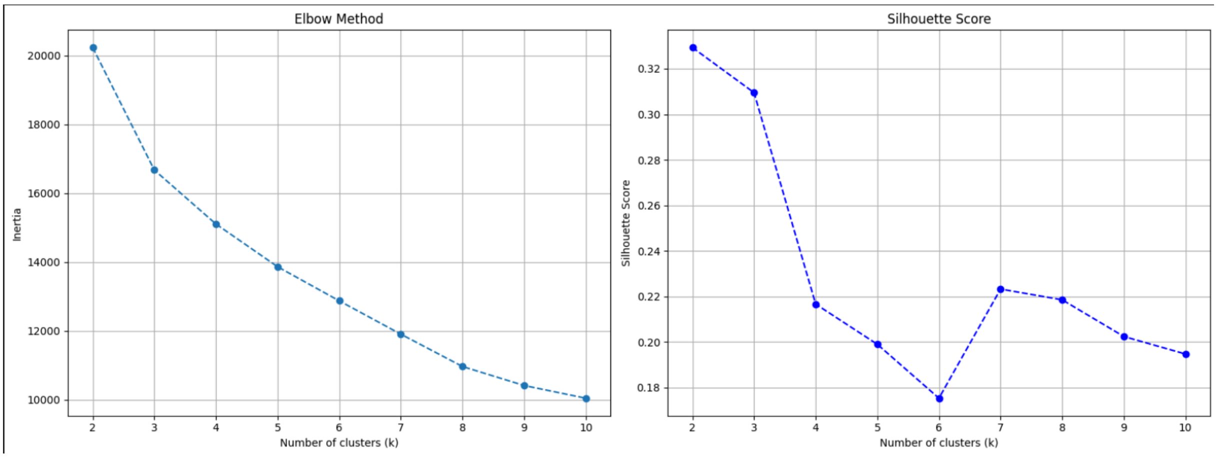
- **Output**

```
data['Cluster'] = cluster_labels

print(f"Total number of clusters formed: {optimal_k}")
print(data.groupby('Cluster').size())

silhouette_avg = silhouette_score(X_scaled, cluster_labels)
print(f"Silhouette Score for k={optimal_k}: {silhouette_avg}")
```

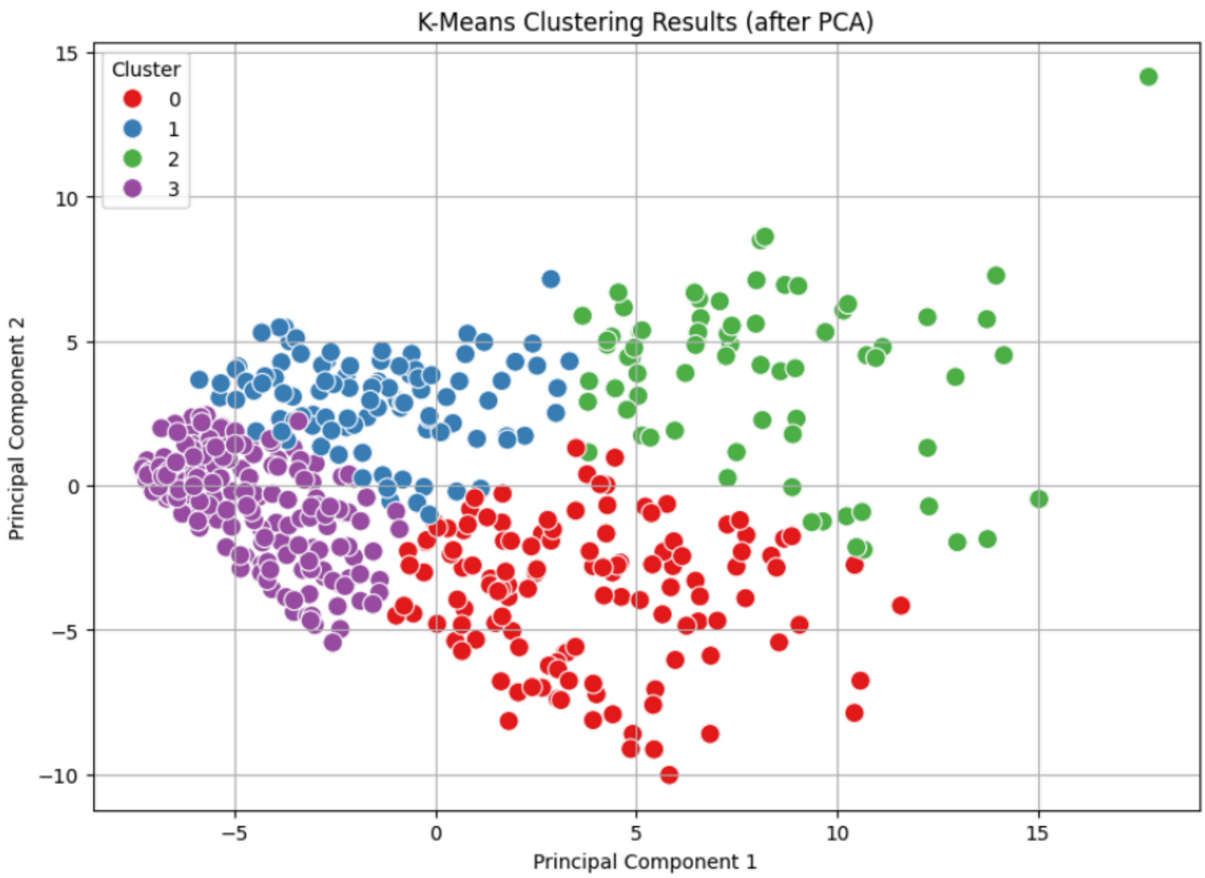
- Displays the number of samples in each cluster.
- Shows the Silhouette Score for the selected k to evaluate clustering quality.

IV. Results

- The results are saved in the file:
    - Kmeans.png:



    - Kmeans_PCA.png:



Lesson 4

I. 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
- Propose a method for estimating player values. How do you select feature and model?

II. Required Libraries

- **pandas:** Data manipulation and analysis
- **selenium:** Web automation to interact with websites
- **webdriver_manager:** Automatically handles ChromeDriver setup
- **time:** Manage time delays
- **sklearn.model_selection**: Data splitting, hyperparameter tuning.
- **sklearn.preprocessing**: Data normalization and transformation.
- **sklearn.compose**: Apply preprocessing to feature groups.
- **sklearn.pipeline**: Create pipelines combining preprocessing and modeling.
- **sklearn.metrics**: Model performance evaluation (r^2, MAE).
- **sklearn.impute**: Handling missing data.
- **xgboost**: Machine learning model based on decision trees.
- **warnings**: Managing and suppressing warnings.

III. Code

a. 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

1. Idea

- Filter players whose Minutes played is greater than 900.
- Navigate to the Premier League 2024–2025 page on footballtransfers.com.
- Extract the link to the full list of players.
- Get player data in each page
- Use the website's search function to look up any players not found on the main list.

2. Detailed Description of Key Components

- **Filter Players with > 900 Minutes**

```
players_with_900_minutes = df[df['Minutes'] > 900]['Name'].str.strip().str.lower().tolist()
print("Number of players with Minutes > 900:", len(players_with_900_minutes))
```

- Generate a lowercase list of players who have played over 900 minutes.

- **Loop Through 22 Pages and Extract ETV**

```python
for i in range(1, 23):
    paged_url = href if i == 1 else f"{href}/{i}"
    print(f"\n🔎 Processing page: {paged_url}")

    retry_count = 0
    while retry_count < 3:
        try:
            driver.get(paged_url)
            time.sleep(10 if i == 1 else 3)

            wait.until(lambda d: len(d.find_elements(By.CSS_SELECTOR, 'td.td-player')) > 0)

            table = driver.find_element(By.CSS_SELECTOR, 'table.mvp-table')
            tbody = table.find_element(By.TAG_NAME, 'tbody')
            rows = tbody.find_elements(By.TAG_NAME, 'tr')

            invalid_count = 0
            for row in rows:
                try:
                    player_td = row.find_element(By.CSS_SELECTOR, 'td.td-player')
                    name = player_td.text.strip().split('\n')[0]
                    name_key = name.lower()
                    if name_key in players_with_900_minutes and name not in found_players:
                        value_td = row.find_element(By.CSS_SELECTOR, 'span.player-tag')
                        transfer_value = value_td.text

                        results.append({"Name": name, "ETV": transfer_value})
                        found_players.add(name)
                except:
                    invalid_count += 1

            if invalid_count:
                print(f"⚠ Skipped {invalid_count} invalid rows.")
            break

        except Exception as e:
```

- Loops through up to 22 pages to search for players and extract their transfer value.
    + WebDriverWait to wait for elements.
    + find_elements(By.TAG_NAME, 'tr') to locate table rows.
    + set() to prevent duplicate players

- **Search for Missing Players One by One**

```python
missing_players = original_set - found_set

print(f"\n❌ Number of players NOT found on the main page: {len(missing_players)}")
for missing in sorted(missing_players):
    print("⛔", missing.title())

for player in missing_players:
    print(f"🔎 Searching for: {player}")
    try:
        driver.get("https://www.footballtransfers.com")
        time.sleep(3)

        search_icon = wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, "div.fa.fa-search")))
        search_icon.click()

        search_panel = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "div.search-panel")))

        search_box = search_panel.find_element(By.CSS_SELECTOR, "input")
        search_box.clear()
        search_box.send_keys(player)
        time.sleep(2)
        search_box.send_keys(Keys.RETURN)

        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "div.searchResults a")))
        first_result = driver.find_element(By.CSS_SELECTOR, "div.searchResults a")

        player_name_elem = first_result.find_element(By.CSS_SELECTOR, "div.text b")
        player_name = player_name_elem.text.strip()

        wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "div.pl_value")))
        etv_value = driver.find_element(By.CSS_SELECTOR, "div.pl_value").text.strip()

        results.append({"Name": player_name, "ETV": etv_value})

    except Exception as e:
        print(f"❌ Could not retrieve data for {player}: {e}")
```

- Opens the search panel and enters missing player names to retrieve their ETVs.

- **Export Results to CSV**

```python
output_df = pd.DataFrame(results)
output_df.to_csv("results4.csv", index=True, encoding="utf-8-sig")
print("\n✅ Results saved to 'results4.csv'")
```

- Saves final list of players and their transfer values to results4.csv.

b. Propose a method for estimating player values. How do you select feature and model?

1. **Idea**

- Handling missing values by using KNN imputation for features like "Age," "Goals," and "Assists."
- Creating a new feature, "Goals_x_Assists," which is the product of "Goals" and "Assists."
- Using the XGBoost regressor (a gradient boosting algorithm) to predict the ETV.
- Evaluating the model's performance using R^2 score and Mean Absolute Error (MAE).

2. **Detailed Description of Key Components**

- **Data Preprocessing**

```
def load_and_preprocess_data(file_path):
    df = pd.read_csv(file_path)

    df["Age"] = df["Age"].apply(lambda x: int(str(x).split("-")[0]) if pd.notna(x) and "-" in str(x) else np.nan)

    imputer = KNNImputer(n_neighbors=3)
    df[["Age", "Goals", "Assists"]] = imputer.fit_transform(df[["Age", "Goals", "Assists"]])
    df["Age"] = df["Age"].astype(int)

    df["Goals_x_Assists"] = df["Goals"] * df["Assists"]

    return df
```

- The data is loaded from a CSV file into a DataFrame `df`.
- If the "Age" value is in the format "n-m" (years-months), the first part (`n`) is taken as the age. Otherwise, it is set to NaN.
- The KNN algorithm is used to impute missing values in the "Age", "Goals", and "Assists" columns by predicting the missing values based on the nearest neighbors.
- The "Goals_x_Assists" feature is created by multiplying "Goals" and "Assists".

- **Data Splitting and Pipeline Creation**

```
X = df.drop(columns=["ETV", "Name"])
y = df["ETV"].str.replace("€", "").str.replace("M", "").astype(float)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- The features are divided into two groups: numeric features (numeric_features) and categorical features (categorical_features).
- The numeric features are standardized using StandardScaler, and categorical features are encoded using OneHotEncoder. This step ensures the model can handle different types of data appropriately.

- **Model Creation and Training**

```python
model = Pipeline([
    ("preprocessor", preprocessor),
    ("regressor", XGBRegressor(random_state=42))
])
```

- A pipeline is created combining preprocessing steps and the model (XGBRegressor). This makes it easier to apply the preprocessing steps to the data before training the model.

- **Hyperparameter Tuning**

```python
model = Pipeline([
    ("preprocessor", preprocessor),
    ("regressor", XGBRegressor(random_state=42))
])

param_grid = {
    "regressor__n_estimators": [200, 300],
    "regressor__max_depth": [6, 8, 10],
    "regressor__learning_rate": [0.01, 0.1],
    "regressor__subsample": [0.8, 1.0],
    "regressor__colsample_bytree": [0.8, 1.0]
}

grid_search = GridSearchCV(
    model,
    param_grid,
    cv=5,
    scoring="r2",
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

- Grid search is used to optimize the hyperparameters of the XGBRegressor. The parameters being tuned include the number of trees (n_estimators), the maximum depth of the trees (max_depth), learning rate (learning_rate), and others.

- **Model Evaluation**

```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"✅ Best Parameters: {grid_search.best_params_}")
print(f"✅ R² Score: {r2:.4f}")
print(f"✅ MAE: {mae:.2f}")
```

- The best model from grid_search is used to predict the test data, and two evaluation metrics are computed: R^2 score and Mean Absolute Error (MAE).

IV. Result

- The results are saved in the file 'results_bai_4.csv'

	Name	ETV
0	Erling Haaland	€199.6M
1	Alexander Isak	€119.4M
2	Cole Palmer	€117.4M
3	Alexis Mac Allister	€117M
4	Declan Rice	€116.4M
5	Bukayo Saka	€113M
6	Phil Foden	€99.8M
7	Ryan Gravenberch	€85.5M
8	Moisés Caicedo	€82.8M
9	Bruno Guimarães	€82.3M
10	Morgan Rogers	€79.6M
11	William Saliba	€79.5M
12	Omar Marmoush	€79.1M
13	Joško Gvardiol	€77.4M
14	Sávio	€74.8M
15	Rúben Dias	€74.5M
16	Enzo Fernández	€74.2M
17	Lucas Paquetá	€73.5M
18	Archie Gray	€72.5M
19	Dominik Szoboszlai	€70.5M
20	Luis Díaz	€69.3M
21	Cody Gakpo	€67.7M
22	Brennan Johnson	€67M
23	Nicolas Jackson	€66.9M
24	Leny Yoro	€66.2M
25	Murillo	€66M