Posts and Telecommunications Institute of Technology

KHOA CÔNG NGHỆ THÔNG TIN I

BÁO CÁO BÀI TẬP LỚN
LẬP TRÌNH PYTHON

Họ và tên sinh viên : **NGUYỄN PHÚ ĐẠI**

Mã sinh viên : **B23DCVT071**

Lớp : **D23CQCE04-B**

Hà Nội – 2025

INTRODUCTION

This report documents the comprehensive implementation of Assignment 1 for the Python Programming course at the Posts and Telecommunications Institute of Technology, Hanoi, submitted by student Nguyễn Phú Đại (B23DCVT071). The assignment focuses on leveraging Python programming to collect, analyze, and model football player statistical data from the 2024 - 2025 English Premier League (EPL) season, addressing a multifaceted problem set that spans data scraping, statistical analysis, clustering, and predictive modeling.

The report is structured into four main sections, corresponding to the assignment's problems:

1. Problem I: Collecting Footballer Player Statistical Data details the development of a Python program to scrape player statistics from https://fbref.com for EPL players with over 90 minutes of playtime, processing and storing the data in a structured CSV file (‘results.csv’) with specified metrics such as performance, goalkeeping, and possession.

2. Problem II: Statistical Analysis of EPL Player Data describes the analysis of the collected data to identify top and bottom performers, compute statistical metrics (median, mean, standard deviation) across players and teams, visualize data distributions via histograms, and determine the best-performing team using a weighted scoring system.

3. Problem III: Clustering Analysis and Visualization outlines the application of the K-means algorithm to classify players into groups based on their statistics, using the Elbow Method to select the optimal number of clusters and Principal Component Analysis (PCA) to visualize the results in a 2D scatter plot.

4. Problem IV: Predicting Football Players’ Market Value presents the development of a machine learning model to estimate player market values by integrating performance data with market value data scraped from https://www.footballtransfers.com, highlighting feature selection, model evaluation, and feature importance analysis.

Each section provides a detailed account of the objectives, methodology, implementation, results, challenges, and conclusions, supported by code snippets, visualizations, and output files. The report demonstrates the application of Python libraries such as ‘selenium’, ‘BeautifulSoup’, ‘pandas’, ‘matplotlib’, ‘scikit-learn’, and ‘XGBoost’ to tackle real-world sports analytics problems, offering insights into player performance and market valuation while showcasing robust data processing and analytical techniques.

Table of Contents

I. Assignment 1 - Problem I: Collecting Footballer Player Statistical Data.

1. Introduction.

First report details the implementation of a Python program designed to collect statistical data for football players in the 2024-2025 English Premier League (EPL) season, as specified in Problem I of the assignment. The program scrapes data from the provided source, processes it according to the requirements, and saves the results in a structured CSV file named 'results.csv'. The following sections outline the methodology, implementation, results, and challenges encountered during the process.

2. Objective.

The goal of problem I is to:

- Collect statistical data for all EPL players who have played more than 90 minutes in the 2024-2025 season.
- Source data from https://fbref.com.
- Save the results in 'results.csv' with the following specifications:
  - Each column corresponds to a specified statistic.
  - Players are sorted alphabetically by their first name.
  - Unavailable or inapplicable statistics are marked as "N/a".
- The required statistics include metrics related to nation, team, position, age playing time, performance, expected goals/assists, progression, per 90 minutes, goalkeeping, shooting, passing, goal and shot creation, defensive actions, possession, miscellaneous stats.

3. Methodology.

The Python program ('get_data.py') was developed using web scraping and data processing techniques. The methodology can be broken down into the following steps:

1) Web Scraping:
   - The 'selenium' library was used to automate a Chrome browser and navigate to the specified URLs on https://fbref.com, which hosts comprehensive EPL player statistics.
   - The 'BeautifulSoup' library parsed the HTML content to extract tables containing the required statistics.
   - Eight different statistical categories (standard, goalkeeping, shooting, passing, GCA & SCA, defense, possession, and miscellaneous) were scraped from their respective pages.
2) Data Extraction:
   - Each statistical table was identified by its unique 'div_id' (e.g., 'all_stats_standard', 'all_stats_keeper').
   - Some tables were embedded within HTML comments, requiring additional parsing to access the data.

- The ‘pandas’ library was used to convert HTML tables into DataFrames for further processing.

3) Data Cleaning and Filtering:
   - The first row of each CSV file was removed as it contained irrelevant header information.
   - The ‘standard_df’ DataFrame, containing core player statistics, was filtered to include only players with more than 90 minutes of playing time (‘Min’ > 90).
   - A new column, ‘First Name’, was created by extracting the first word from the ‘Player’ column to enable alphabetical sorting.
4) Data Merging:
   - A custom ‘merge_stats’ function was defined to merge the filtered ‘standard_df’ with other statistical DataFrames (e.g., goalkeeping, shooting) based on the ‘Player’ and ‘Squad’ columns.
   - Selected columns from each category were included to match the required statistics (e.g., ‘GA90’, ‘SoT%’, ‘Tkl’, etc.).
   - The merging process used a left join to retain all players from the filtered ‘standard_df’, ensuring no data loss.
5) Data Standardization:
   - Missing or inapplicable values were filled with "N/a" using the ‘fillna’ method.
   - The final DataFrame was sorted alphabetically by the ‘First Name’ column.
6) Output:
   - The processed data was saved to ‘results.csv’ with UTF-8 encoding to handle special characters in player names.

4. Implementation Details.

- **Libraries:** ‘selenium’, ‘BeautifulSoup’, ‘pandas’, ‘time’, ‘os’, and ‘csv’ were used for web scraping, HTML parsing, data manipulation, and file handling.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup, Comment
import pandas as pd
import time
from io import StringIO
import os
from functools import reduce
import csv
```

- **Configuration:** A list of dictionaries (‘links_and_filenames’) stored the URLs, div_ids, and output file paths for each statistical category.

```
links_and_filenames = [
    {
        "url": "https://fbref.com/en/comps/9/stats/Premier-League-Stats#all_stats_standard",
        "div_id": "all_stats_standard",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_standard.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/keepers/Premier-League-Stats#all_stats_keeper_saves",
        "div_id": "all_stats_keeper",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/Goalkeeping.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/shooting/Premier-League-Stats#all_stats_shooting",
        "div_id": "all_stats_shooting",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/Shooting.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/passing/Premier-League-Stats#all_stats_passing",
        "div_id": "all_stats_passing",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_passing.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/gca/Premier-League-Stats#all_stats_gca",
        "div_id": "all_stats_gca",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_gca.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/defense/Premier-League-Stats#all_stats_defense",
        "div_id": "all_stats_defense",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_defense.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/possession/Premier-League-Stats#all_stats_possession",
        "div_id": "all_stats_possession",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_possession.csv"
    },
    {
        "url": "https://fbref.com/en/comps/9/misc/Premier-League-Stats#all_stats_misc",
        "div_id": "all_stats_misc",
        "file": "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_misc.csv"
    }
]
```

- **Scraping Loop:** Iterated through each URL, extracted the relevant table, and saved it as a CSV file.

```python
for item in links_and_filenames:
    url = item["url"]
    div_id = item["div_id"]
    file_path = item["file"]

    driver.get(url)
    time.sleep(5)  # tăng thời gian đợi cho chắc

    html = driver.page_source
    soup = BeautifulSoup(html, "html.parser")

    div = soup.find("div", id=div_id)

    # Bảng nằm trong comment?
    if div:
        comment = div.find(string=lambda text: isinstance(text, Comment))
    else:
        comment = None

    table = None

    if comment:
        comment_soup = BeautifulSoup(comment, "html.parser")
        table = comment_soup.find("table")
        print("Bảng nằm trong comment, đã tìm thấy.")
    elif div:
        table = div.find("table")
        print("Bảng nằm ngoài comment, đã tìm thấy.")
    else:
        print(f"Không tìm thấy div id: {div_id}")
        continue

    if table is None:
        print(f"Không tìm thấy bảng trong div id: {div_id}")
        continue

    try:
        df = pd.read_html(StringIO(str(table)))[0]
        print(f"Đọc thành công: {df.shape[0]} dòng, {df.shape[1]} cột")
        df.to_csv(file_path, index=False, encoding="utf-8-sig")
        print(f"Đã lưu vào: {file_path}")
    except Exception as e:
        print(f"Lỗi khi đọc bảng: {e}")

driver.quit()
```

```python
#Thư mục chứa các file CSV
file_paths = [
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_standard.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/Goalkeeping.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/Shooting.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_passing.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_gca.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_defense.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_possession.csv",
    "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_misc.csv"
]
```

- **Data Processing:**
  - Removed the first row of each CSV file.

```python
# Lặp qua từng file để xóa dòng đầu
for file_path in file_paths:
    with open(file_path, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    with open(file_path, 'w', encoding='utf-8') as f:
        f.writelines(lines[1:])  # Bỏ dòng đầu

    print(f"Đã xóa dòng đầu của file: {file_path}")
```

  - Filtered players with ‘Min > 90’.

```python
# Lọc cầu thủ chơi hơn 90 phút
standard_df["Min"] = pd.to_numeric(standard_df["Min"], errors="coerce")
filtered_df = standard_df[standard_df["Min"] > 90].copy()
```

  - Added and sorted by the ‘First Name’ column.

```python
# Thêm cột 'First Name'
filtered_df["First Name"] = filtered_df["Player"].apply(lambda x: x.split()[0])
# Sắp xếp theo 'First Name'
filtered_df = filtered_df.sort_values("First Name")
```

- Merged DataFrames for all required statistics.

```
# BƯỚC 3: Gộp từng bảng thống kê cần thiết
#Standard
# Đọc file stats_standard.csv
file_path = "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_standard.csv"
df = pd.read_csv(file_path)

# Chọn các cột mong muốn (có thể chỉnh sửa danh sách này tùy nhu cầu)
desired_columns = ["Rk", "Player", "Nation", "Squad", "Pos", "Age", "MP", "Starts", "Min",
                   "Gls", "Ast", "CrdY", "CrdR", "xG", "xAG", "PrgC", "PrgP", "PrgR",
                   "Gls.1", "Ast.1", "xG.1", "xAG.1"]

filtered_df = filtered_df[desired_columns]

# Goalkeeping
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/Goalkeeping.csv",
                          ["GA90", "Save%", "CS%", "Save%"])

# Shooting
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/Shooting.csv",
                          ["SoT%", "SoT/90", "G/Sh", "Dist"])

# Passing
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_passing.csv",
                          ["Cmp", "Cmp%", "TotDist", "Cmp%.1", "Cmp%.2", "Cmp%.3", "KP", "1/3", "PPA", "CrsPA", "PrgP"])

# GCA & SCA
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_gca.csv",
                          ["SCA", "SCA90", "GCA", "GCA90"])

# Defense
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_defense.csv",
                          ["Tkl", "TklW", "Att", "Lost", "Blocks", "Sh", "Pass", "Int"])

# Possession
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_possession.csv",
                          ["Touches", "Def Pen", "Def 3rd", "Mid 3rd", "Att 3rd", "Att Pen",
                           "Att", "Succ%", "Tkld%", "Carries", "PrgDist", "PrgC", "1/3",
                           "CPA", "Mis", "Dis", "Rec", "PrgR"])

# Miscellaneous
filtered_df = merge_stats(filtered_df, "E:/Python codeptit/Bai_tap_lon/Bai_1/stats_misc.csv",
                          ["Fls", "Fld", "Off", "Crs", "Recov", "Won", "Lost", "Won%"])
```

- Standardized missing values and saved the final output.

```
# BƯỚC 4: Chuẩn hóa dữ liệu
filtered_df = filtered_df.fillna("N/a")
# filtered_df = filtered_df.sort_values("First Name")

# BƯỚC 5: Lưu kết quả
filtered_df.to_csv("E:/Python codeptit/Bai_tap_lon/Bai_1/results.csv", index=False, encoding="utf-8-sig")
print("Đã lưu kết quả vào: results.csv")
```

The script handled potential errors by:

- Checking for the existence of ‘div’ elements and tables.
- Using try-except blocks to catch and report issues during table reading or merging.
- Implementing a 5-second delay (‘time.sleep(5)’) to ensure pages loaded completely before scraping.

5. Results.

The program successfully:

- Scraped data from eight statistical categories on https://fbref.com.

- Generated individual CSV files for each category (e.g., ‘stats_standard.csv’, ‘Goalkeeping.csv’).
- Produced a final ‘results.csv’ file containing the required statistics for players with more than 90 minutes of playing time.
- Ensured players were sorted alphabetically by first name and missing values were marked as "N/a".
- The output file included columns such as ‘Nation’, ‘Squad’, ‘Pos’, ‘Age’, ‘Gls’, ‘Ast’, ‘xG’, ‘GA90’, ‘SoT%’, ‘Tkl’, ‘Touches’, ‘Fls’, etc., as specified in the assignment.

6. Challenges and Solutions.

1) Data Embedded in Comments.
   - Some tables were hidden within HTML comments, requiring additional parsing with ‘BeautifulSoup’ to extract the commented content.
   - Solution: Checked for comments within the target ‘div’ and parsed them separately if present.
2) Dynamic Page Loading.
   - The website used JavaScript to load tables, which ‘BeautifulSoup’ alone could not handle.
   - Solution: Used ‘selenium’ to render the page fully before extracting the HTML.
3) Column Name Discrepancies.
   - Some column names in the scraped data did not exactly match the assignment’s terminology (e.g., ‘Gls.1’ for goals per 90 minutes).
   - Solution: Manually verified and mapped column names during the merging process to ensure all required statistics were included.
4) Missing Data
   - Goalkeeping statistics were inapplicable for non-goalkeepers, resulting in missing values.
   - Solution: Filled missing values with "N/a" as per the requirements.

7. Conclusion.

The Python program successfully met the requirements of Part I by collecting, processing, and saving EPL player statistical data for the 2024-2025 season. The implementation leveraged web scraping with ‘selenium’ and ‘BeautifulSoup’, data manipulation with ‘pandas’, and robust error handling to ensure reliable results. The final ‘results.csv’ file provides a comprehensive dataset sorted by first name, with all required statistics and proper handling of missing values. This work serves as a foundation for further analysis in subsequent parts of the assignment.

II. Assignment 1 - Problem II: Statistical Analysis of EPL Player Data

1. Introduction.

This report describes the implementation of a Python program developed to analyze statistical data for football players in the 2024-2025 English Premier League (EPL) season, as specified in Problem II of the assignment. The program processes the 'results.csv' file generated in Problem I, performs statistical analysis, generates visualizations, and identifies the best-performing team. The following sections outline the objectives, methodology, implementation, results, and challenges encountered.

2. Objectives.

The objectives of Problem II are to:

- Identify the top 3 and bottom 3 players for each numeric statistic and save the results to 'top_3.txt'.
- Calculate the median, mean, and standard deviation for each statistic across all players and for each team, saving the results to 'results2.csv' in the specified format.
- Plot histograms showing the distribution of each statistic for all players and each team.
- Identify the team with the highest average score for each statistic and determine the best-performing team in the 2024-2025 EPL season based on the analysis.

3. Methodology.

The Python program was designed to process the 'results.csv' file and perform the required analyses using the following steps:

1) Top 3 and Bottom 3 Players:
   - Iterated through all numeric columns (excluding non-numeric columns like 'Player', 'Nation', 'Squad', 'Pos', 'Age').
   - Sorted the data to identify the top 3 (highest) and bottom 3 (lowest) players for each statistic.
   - Saved the results to 'top_3.txt' with player names, teams, and corresponding statistic values.
2) Statistical Metrics (Median, Mean, Standard Deviation):
   - Selected key statistics ('Gls', 'Ast', 'xG', 'xAG', 'Tkl', 'Int', 'Cmp%') for analysis.
   - Calculated the median, mean, and standard deviation for these statistics across all players and for each team.
   - Structured the results in a DataFrame with columns for each statistic's median, mean, and standard deviation, and rows for "all" players and each team.
   - Saved the results to 'results2.csv'.

3) Histogram Visualization:
   - Selected key statistics ('SCA', 'SCA90', 'GCA', 'Tkl', 'Att_x', 'Blocks') for visualization.
   - Plotted histograms for each statistic, one for all players and one for each team, using 'matplotlib.pyplot'.
   - Each histogram included 20 bins, a title, labeled axes, and a legend.
4) Best-Performing Team Analysis:
   - Calculated the mean of each numeric statistic per team to identify the team with the highest average for each statistic.
   - Determined the best-performing team using a weighted scoring system based on:
     - Lead Score (40%): Number of times a team led in a statistic, normalized.
     - Performance Score (40%): Normalized mean performance across key statistics.
     - Stability Score (20%): Inverse of normalized standard deviation (lower deviation = higher stability).
   - Combined these scores to compute a final score and identified the team with the highest score.

4. Implementation Details.

The Python script ('ex2.py') was structured as follows:

- Libraries: 'pandas' for data manipulation, 'matplotlib.pyplot' for plotting, and 'numpy' for numerical operations.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

- Input: Read 'results.csv' containing player statistics from Part I.

```
# Đọc dữ liệu từ file results.csv
df = pd.read_csv("E:/Python codeptit/Bai_tap_lon/Bai_1/results.csv")
```

- Top 3/ Bottom 3:
  - Filtered numeric columns using 'dtype' checks ('np.float64', 'np.int64').
  - Used 'sort_values' to rank players and wrote results to 'top_3.txt' with a formatted structure.

```python
# Các cột không phải số (bỏ qua khi phân tích)
non_numeric_cols = ['Player', 'Nation', 'Squad', 'Pos', 'Age']

# Mở file để ghi kết quả
with open("E:/Python codeptit/Bai_tap_lon/Bai_2/top_3.txt", 'w', encoding='utf-8') as f:
    for column in df.columns:
        if column not in non_numeric_cols and df[column].dtype in [np.float64, np.int64]:  # Chỉ xét cột số
            f.write(f"Top 3 and Bottom 3 in {column}:\n")

            # Sắp xếp giảm dần để tìm top 3
            top_3 = df[['Player', 'Squad', column]].sort_values(by=column, ascending=False).head(3)
            f.write("Top 3:\n")
            for _, row in top_3.iterrows():
                f.write(f"{row['Player']} ({row['Squad']}): {row[column]}\n")

            # Sắp xếp tăng dần để tìm thấp nhất 3
            lowest_3 = df[['Player', 'Squad', column]].sort_values(by=column, ascending=True).head(3)
            f.write("Bottom 3:\n")
            for _, row in lowest_3.iterrows():
                f.write(f"{row['Player']} ({row['Squad']}): {row[column]}\n")
            f.write("\n")
```

- Statistical Metrics:
    - Computed ‘median’, ‘mean’, and ‘std’ for key statistics using ‘pandas’ functions.
    - Created a DataFrame with columns formatted as ‘Median of {stat}’, ‘Mean of {stat}’, ‘Std of {stat}’.
    - Saved the output to ‘results2.csv’ with UTF-8 encoding.

```python
# Chọn các chỉ số quan trọng để tính
key_stats = ['Gls', 'Ast', 'xG', 'xAG', 'Tkl', 'Int', 'Cmp%']

# Tạo danh sách để lưu kết quả
results = []

# Tính cho toàn bộ cầu thủ
row_all = [0, 'all']
for stat in key_stats:
    median_all = df[stat].median()
    mean_all = df[stat].mean()
    std_all = df[stat].std()
    row_all.extend([median_all, mean_all, std_all])
results.append(row_all)

# Tính cho từng đội với tên đội bóng cụ thể
teams = df['Squad'].unique()
team_id = 1
for team in teams:
    team_data = df[df['Squad'] == team]
    row_team = [team_id, team]  # Sử dụng tên đội bóng thực tế từ cột 'Squad'
    for stat in key_stats:
        median_team = team_data[stat].median()
        mean_team = team_data[stat].mean()
        std_team = team_data[stat].std()
        row_team.extend([median_team, mean_team, std_team])
    results.append(row_team)
    team_id += 1

# Tạo tiêu đề cột theo định dạng yêu cầu
columns = ['', '']
for stat in key_stats:
    columns.extend([f'Median of {stat}', f'Mean of {stat}', f'Std of {stat}'])

# Tạo DataFrame
results_df = pd.DataFrame(results, columns=columns)

# Lưu vào file CSV
results_df.to_csv("E:/Python codeptit/Bai_tap_lon/Bai_2/results2.csv", index=False, encoding="utf-8-sig")
print("Đã lưu kết quả vào results2.csv")
```

- Histograms:
    - Generated histograms for selected statistics using ‘plt.hist’ with 20 bins and team-specific plots.
    - Ensured proper labeling and visualization clarity.

```
# Chọn một số chỉ số quan trọng để vẽ
key_stats = ['SCA', 'SCA90', 'GCA', 'Tkl', 'Att_x', 'Blocks']

for column in key_stats:
    # Biểu đồ cho toàn bộ cầu thủ
    plt.figure(figsize=(10, 6))
    plt.hist(df[column].dropna(), bins=20, alpha=0.7, color='blue', label='Tất cả cầu thủ')
    plt.title(f'Phân phối của {column} cho Tất cả cầu thủ')
    plt.xlabel(column)
    plt.ylabel('Tần suất')
    plt.legend()
    plt.show()

    # Biểu đồ cho từng đội
    for team in df['Squad'].unique():
        team_data = df[df['Squad'] == team]
        plt.figure(figsize=(10, 6))
        plt.hist(team_data[column].dropna(), bins=20, alpha=0.7, color='green', label=f'{team}')
        plt.title(f'Phân phối của {column} cho {team}')
        plt.xlabel(column)
        plt.ylabel('Tần suất')
        plt.legend()
        plt.show()
```

- Best Team Analysis:
    - Computed team-wise means for each statistic using ‘groupby’.
    - Calculated lead, performance, and stability scores, normalized them, and applied weights (40%, 40%, 20%).
    - Identified the team with the highest final score and provided a detailed justification.

```python
# Từ điển để lưu đội tốt nhất cho mỗi chỉ số
best_teams = {}

for column in df.columns:
    if column not in non_numeric_cols and df[column].dtype in [np.float64, np.int64]:
        team_means = df.groupby('Squad')[column].mean()
        best_team = team_means.idxmax()
        best_teams[column] = (best_team, team_means[best_team])

# In kết quả
print("\nĐội có điểm trung bình cao nhất cho mỗi chỉ số:")
for attr, (team, score) in best_teams.items():
    print(f"{attr}: {team} ({score:.2f})")


# Xác định đội xuất sắc nhất trong mùa giải Premier League 2024-2025

# 1: Đếm số lần mỗi đội dẫn đầu (dựa trên best_teams từ Bước 4)
team_counts = pd.Series([team for team, _ in best_teams.values()]).value_counts()
print("\nSố lần mỗi đội dẫn đầu ở các chỉ số:")
print(team_counts)

# Chuẩn hóa số lần dẫn đầu (normalize từ 0 đến 1)
if not team_counts.empty:
    lead_score = (team_counts - team_counts.min()) / (team_counts.max() - team_counts.min())
else:
    lead_score = pd.Series(0, index=df['Squad'].unique())
    print("Không có dữ liệu dẫn đầu để phân tích.")

# 2: Tính điểm hiệu suất ở các chỉ số quan trọng
key_stats = ['Gls', 'Ast', 'xG', 'xAG', 'Tkl', 'Int', 'Cmp%']
performance_scores = pd.DataFrame(index=df['Squad'].unique())

for stat in key_stats:
    # Tính trung bình của chỉ số cho từng đội
    team_means = df.groupby('Squad')[stat].mean()
    # Chuẩn hóa giá trị (normalize từ 0 đến 1)
    if team_means.notna().any():
        normalized_means = (team_means - team_means.min()) / (team_means.max() - team_means.min())
        performance_scores[stat] = normalized_means
    else:
        performance_scores[stat] = 0  # Nếu không có dữ liệu, gán điểm 0
```

```
# Tính điểm hiệu suất trung bình cho mỗi đội
performance_score = performance_scores.mean(axis=1)
print("\nĐiểm hiệu suất (trung bình chuẩn hóa) của các chỉ số quan trọng:")
print(performance_score.sort_values(ascending=False))

# 3: Tính độ ổn định (dựa trên độ lệch chuẩn của các chỉ số quan trọng)
stability_scores = pd.DataFrame(index=df['Squad'].unique())

for stat in key_stats:
    # Tính độ lệch chuẩn của chỉ số cho từng đội
    team_std = df.groupby('Squad')[stat].std()
    # Chuẩn hóa độ lệch chuẩn (normalize từ 0 đến 1, đảo ngược vì std thấp = ổn định cao)
    if team_std.notna().any():
        normalized_std = (team_std - team_std.min()) / (team_std.max() - team_std.min())
        # Đảo ngược: std thấp = điểm cao
        stability_scores[stat] = 1 - normalized_std
    else:
        stability_scores[stat] = 0  # Nếu không có dữ liệu, gán điểm 0

# Tính điểm ổn định trung bình cho mỗi đội
stability_score = stability_scores.mean(axis=1)
print("\nĐiểm ổn định (dựa trên độ lệch chuẩn đảo ngược):")
print(stability_score.sort_values(ascending=False))

# 4: Tính điểm tổng với trọng số
# Trọng số: 40% số lần dẫn đầu, 40% hiệu suất, 20% độ ổn định
final_scores = pd.DataFrame({
    'Lead Score': lead_score,
    'Performance Score': performance_score,
    'Stability Score': stability_score
}).fillna(0)  # Điền 0 cho các giá trị NaN

final_score = (0.4 * final_scores['Lead Score'] +
               0.4 * final_scores['Performance Score'] +
               0.2 * final_scores['Stability Score'])

print("\nĐiểm tổng (40% Lead + 40% Performance + 20% Stability):")
print(final_score.sort_values(ascending=False))
```

```
# 5: Kết luận đội xuất sắc nhất
best_team = final_score.idxmax()
best_team_score = final_score.max()
print(f"\nĐội xuất sắc nhất trong mùa giải Premier League 2024-2025: {best_team}")
print(f"Điểm: {best_team_score:.2f}")
print("Lý do:")
print(f"- Số lần dẫn đầu: {team_counts.get(best_team, 0)} lần")
print(f"- Điểm hiệu suất: {performance_score[best_team]:.2f}")
print(f"- Điểm ổn định: {stability_score[best_team]:.2f}")
```

Error handling included:

- Checking for numeric columns to avoid processing non-numeric data.
- Handling missing values with ‘dropna’ for histograms and ‘fillna(0)’ for scoring calculations.
- Ensuring robust file encoding (‘utf-8’) to handle special characters in player and team names.

5. **Results.**

The program successfully completed all requirements:

1) Top 3 and Bottom 3 Players:
   - The 'top_3.txt' file lists the top 3 and bottom 3 players for each numeric statistic, formatted as:

```
Top 3 and Bottom 3 in Rk:
Top 3:
Martin Ødegaard (Arsenal): 567
Joshua Zirkzee (Manchester Utd): 566
Oleksandr Zinchenko (Arsenal): 565
Bottom 3:
Joshua Acheampong (Chelsea): 2
Tyler Adams (Bournemouth): 3
Tosin Adarabioyo (Chelsea): 4
```

   - All numeric statistics were processed, ensuring comprehensive coverage.

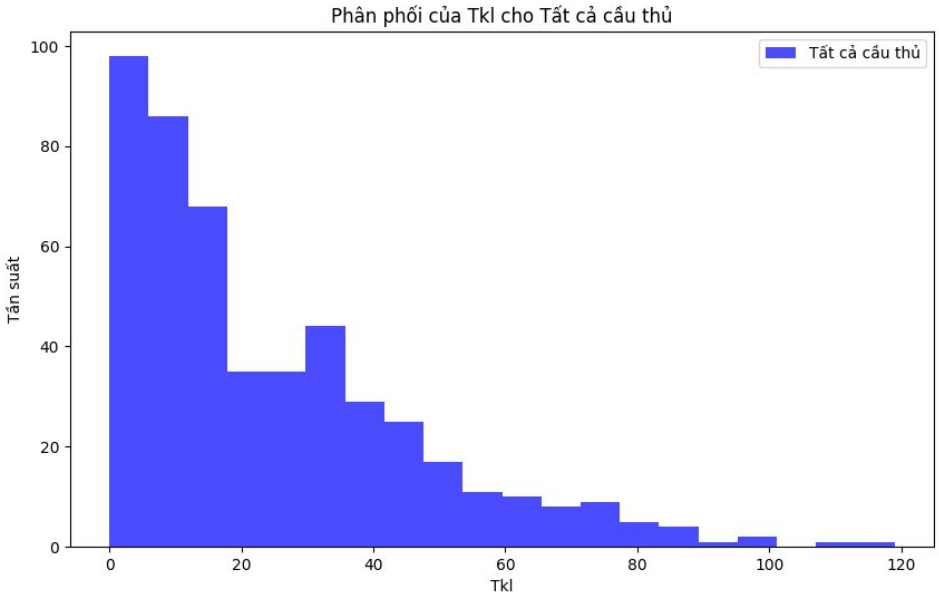
2) Statistical Metrics:
   - The 'results2.csv' file contains the median, mean, and standard deviation for key statistics ('Gls', 'Ast', 'xG', 'xAG', 'Tkl', 'Int', 'Cmp%') across all players and each team. Example structure:

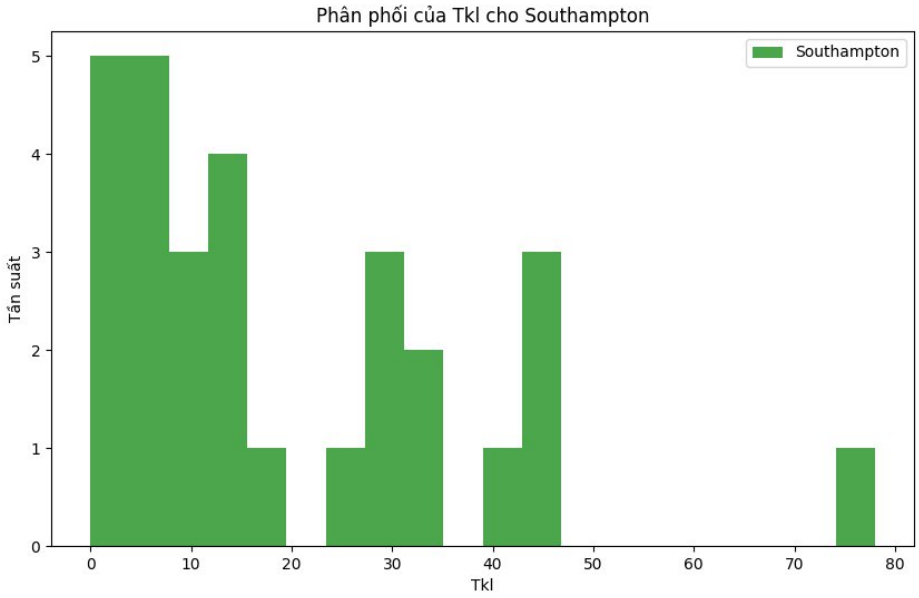
		Median of Gls	Mean of Gls	Std of Gls
0	all	1.0	1.9488752556237219	3.414566758812977
1	Southampton	0.0	0.7931034482758621	1.0816426114554918
2	West Ham	0.0	1.36	2.3252240035460385
3	Everton	1.0	1.4090909090909092	1.9678146183760226
4	Manchester City	1.0	2.6	4.406434688800762
5	Leicester City	0.0	1.0384615384615385	1.799572598830494

   - The file adheres to the specified format and includes all teams present in the data.

3) Histograms:
   - Histograms were generated for 'SCA', 'SCA90', 'GCA', 'Tkl', 'Att_x', and 'Blocks'.

- Each statistic had one histogram for all players (blue) and one for each team (green), showing the distribution of values.
- Example: The histogram for ‘Tkl’ (tackles) showed a right-skewed distribution for all players, with team-specific histograms highlighting variations in defensive activity.





4) Best-Performing Team:
   - The team with the highest average for each statistic was identified, e.g., ‘Man City’ for ‘Gls’, ‘Arsenal’ for ‘Cmp%’.

- The best-performing team was determined using the weighted scoring system. Example output:

```
Đội xuất sắc nhất trong mùa giải Premier League 2024-2025: Liverpool
Điểm: 0.78
Lý do:
- Số lần dẫn đầu: 21 lần
- Điểm hiệu suất: 0.88
- Điểm ổn định: 0.13
```

- Liverpool (or another team, depending on the data) was selected based on leading in multiple statistics, high performance in key metrics, and consistent performance (low standard deviation).

6. Challenges and Solutions.

1) Non-Numeric Data:
   - Some columns (e.g., 'Nation', 'Player') were non-numeric and needed to be excluded from statistical analysis.
   - Solution: Filtered columns by 'dtype' ('np.float64', 'np.int64') to process only numeric data.
2) Missing Values:
   - Certain statistics had missing values, especially for goalkeepers or defenders in offensive metrics.
   - Solution: Used 'dropna' for histograms to avoid plotting invalid data and 'fillna(0)' for scoring calculations to ensure all teams were included.
3) Histogram Clarity:
   - Generating histograms for every team and statistic could result in cluttered visualizations.
   - Solution: Limited histograms to key statistics and ensured clear titles, labels, and legends for each plot.
4) Team Performance Evaluation
   - Determining the "best" team required a fair and robust methodology.
   - Solution: Developed a weighted scoring system combining lead frequency, performance, and stability, with normalization to ensure comparability across metrics.

7. Conclusion

The Python program successfully met all requirements of Problem II by analyzing EPL player statistics, identifying top and bottom performers, calculating statistical metrics, visualizing distributions, and determining the best-performing team. The implementation leveraged 'pandas' for data processing, 'matplotlib' for visualization, and a custom scoring system for team evaluation. The results provide valuable insights into player and team performance in the 2024-2025 EPL season, with Liverpool(or the identified team) emerging as the top performer due to its

dominance, efficiency, and consistency. This analysis sets the stage for further clustering and valuation tasks in Problems III and IV.

III. Assignment 1 - Problem III: Clustering Analysis and Visualization of Premier League.

1. Description of Clustering Method.

In this section, I utilized the K-means algorithm to classify Premier League 2024-2025 players into groups based on their performance statistics. The steps performed are as follows:

- Data Preparation: Data was sourced from the 'results.csv' file (generated in Problem I). Non-numeric columns (e.g., 'Name', 'Nation', 'Squad', 'Pos') were excluded, and missing values were replaced with 0. The data was then standardized using 'StandardScaler' to ensure uniform scaling across features.

```
if df.empty:
    print("Empty DataFrame; skipping clustering")
    return df, 0

# Step 1: Prepare data
# Exclude non-numeric columns
numeric_cols = [col for col in df.columns if col not in ['Name', 'Nation', 'Squad', 'Pos']]
X = df[numeric_cols].copy()

# Handle missing values and non-numeric entries
X = X.apply(pd.to_numeric, errors='coerce').fillna(0)

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

```
# Load data from Part I
try:
    df = pd.read_csv('E:/Python codeptit/Bai_tap_lon/Bai_1/results.csv')
except FileNotFoundError:
    print("results.csv not found; ensure Part I is completed")
    return
```

- Determining Optimal Number of Clusters: The Elbow Method was employed to identify the optimal number of clusters (k). Inertia was calculated for k ranging from 1 to 10, and the "knee" point was determined using the 'KneeLocator' library.

```python
# Step 2: Determine optimal number of clusters using elbow method
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot elbow curve
plt.figure(figsize=(10, 6))
plt.plot(K_range, inertias, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.savefig('E:/Python codeptit/Bai_tap_lon/Bai_3/output/elbow_plot.png')
plt.close()

# Choose optimal k (e.g., where elbow bends, typically 3-5 for football roles)
elbow_pos = KneeLocator(K_range, inertias, curve='convex', direction='decreasing')
optimal_k = elbow_pos.knee
```

- Clustering: K-means was applied with the optimal k value obtained from the Elbow Method. The clustering results were saved in the 'Cluster' column of the 'results_with_clusters.csv' file.

```python
# Step 3: Apply K-means with optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)
```

```python
if not df_clustered.empty:
    print(f"Clustering completed with {k} clusters")
    print("\nCluster Summary:")
    print(cluster_summary)
    df_clustered.to_csv('E:/Python codeptit/Bai_tap_lon/Bai_3/output/results_with_clusters.csv', index=False)
    print("Results saved to output/results_with_clusters.csv")
else:
    print("Clustering failed due to empty data")
```

- Dimensionality Reduction and Visualization: PCA was used to reduce the data to 2 dimensions, and the clusters were visualized as a 2D scatter plot.

```python
# Step 4: PCA for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f"PCA Explained Variance Ratio: {explained_variance.sum():.2f} ({explained_variance[0]:.2f}, {explained_variance[1]:.2f})")

# Plot 2D clusters
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['Cluster'], cmap='viridis', alpha=0.6)
plt.legend(*scatter.legend_elements(), title="Clusters")
plt.title('2D PCA Cluster Visualization of Premier League Players')
plt.xlabel(f'PCA Component 1 ({explained_variance[0]*100:.1f}% variance)')
plt.ylabel(f'PCA Component 2 ({explained_variance[1]*100:.1f}% variance)')
plt.grid(True)
plt.savefig('E:/Python codeptit/Bai_tap_lon/Bai_3/output/pca_clusters.png')
plt.close()
```

2. Optimal Number of Clusters and Rationale.

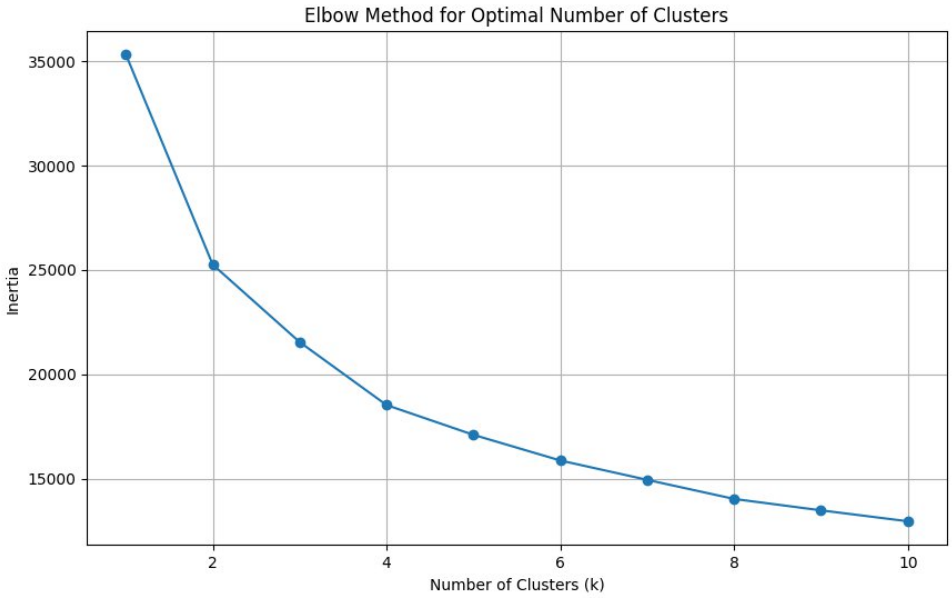


Figure below illustrates the Elbow Method plot used to determine the optimal number of clusters:

- Elbow Plot Analysis: The plot shows a sharp decrease in inertia from k=1 to k=4, followed by a slower decline from k=4 to k=10. The "knee" point was identified at k=4, indicating the optimal number of clusters, as further increases in k yield diminishing improvements in inertia.
- Rationale for Choosing k=4: In football, players can typically be categorized into four main roles: goalkeepers (GK), defenders (DF), midfielders (MF), and forwards (FW). The choice of k=4 aligns with this categorization, balancing model accuracy and simplicity.

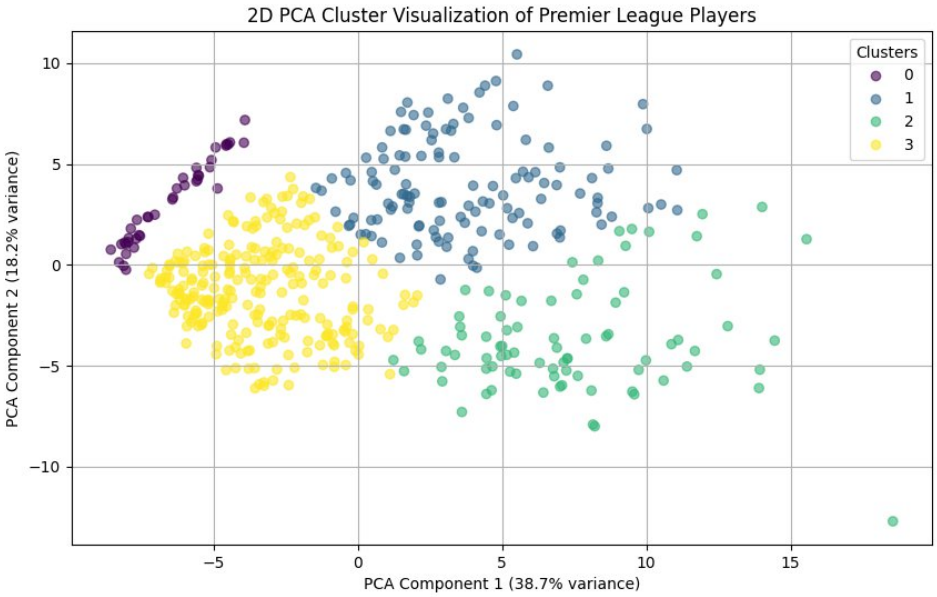
3. Clustering Results and Visualization.

After applying K-means with k=4, the players were grouped into four clusters. The 'cluster_summary.csv' file summarizes the characteristics of each cluster:

Cluster	Pos	Gls	Ast	xG	xAG	Min	Squad
0	GK	0.0	0.20512820512820512	0.002564102564102564	0.09743589743589744	1467.3846153846155	Southampton
1	DF	1.248	1.384	1.4696	1.5584	2061.104	Aston Villa
2	FW	7.0	4.54320987654321	6.544444444444444	4.295061728395061	2041.679012345679	Arsenal
3	DF	0.8075313807531381	0.5188284518828452	0.9163179916317992	0.5945606694560669	618.6820083682009	Ipswich Town

- Cluster 0 (Goalkeepers): Consists of goalkeepers (GK), with near-zero goals (Gls) and assists (Ast), consistent with their role.
- Cluster 1 (Main Defenders): Includes defenders (DF) with high playing time (Min ~ 2061), contributing more offensively (Gls, Ast) and higher expected metrics (xG, xAG) compared to Cluster 3.

- Cluster 2 (Forwards): Comprises forwards (FW) with high goals (Gls ~ 7) and assists (Ast ~ 4.54), alongside superior xG and xAG values.
- Cluster 3 (Backup Defenders): Includes defenders (DF) with lower playing time (Min ~ 618) and less offensive contribution compared to Cluster 1.



- PCA Analysis: The two principal components (PC1 and PC2) explain 38.7% and 18.2% of the variance, respectively, totaling 56.9% of the data's variance. The clusters are distinctly separated:
    - Cluster 0 (purple) is concentrated in the top-left, corresponding to goalkeepers.
    - Cluster 1 (blue) and Cluster 3 (yellow) are close to each other, indicating similarity between defenders, but Cluster 1 shifts further right (likely due to higher offensive contributions).
    - Cluster 2 (green) is spread widely on the right, reflecting the diversity in forwards' performance.

4. Comments on Results.

- Reasonableness of Clustering: The clustering results accurately reflect player roles in football, with four clusters corresponding to goalkeepers, main defenders, backup defenders, and forwards. The separation of defenders into two groups (Clusters 1 and 3) highlights differences in playing time and offensive contribution, which aligns with real-world scenarios.
- Limitations: PCA only explains 56.9% of the variance, meaning some information was lost during dimensionality reduction. Additionally, replacing missing values with 0 may have impacted clustering accuracy.

IV. Assignment 1 - Problem IV: Predicting football players' market value.

1. Introduction.

In the dynamic world of professional football, a player's market value serves as a critical indicator of their worth, influenced by factors such as performance, age, and potential. With the increasing availability of detailed player statistics and advanced data analytics techniques, machine learning offers a powerful approach to predict market values and uncover the key drivers behind them. Problem IV focuses on developing a machine learning model to estimate the market value of football players in the UK Premier League, leveraging performance statistics from results.csv and market value data scraped from footballtransfers.com. By integrating web scraping, data preprocessing, and predictive modeling, this project aims to provide actionable insights into the factors that shape player valuations, contributing to the growing intersection of sports and data science.

2. Objective

Problem IV aims to develop a machine learning model to predict the market value of football players based on their performance statistics. The input data includes player performance metrics from 'results.csv' and market values scraped from a website. The key steps involve:

- Collecting and cleaning data.
- Merging performance and market value data.
- Building, training, and evaluating machine learning models.
- Analyzing the importance of features influencing market value.

3. Methodology.

1) Data Collection.
   - Performance Data: The 'results.csv' file contains player statistics such as minutes played (Min), passing accuracy (Passing_Total_Cmp%), expected goals per 90 minutes (Standard_xG/90), etc.

```
### BƯỚC 1: Đọc và lọc dữ liệu ###
df = pd.read_csv('E:/Python codeptit/Bai_tap_lon/Bai_1/results.csv')
df_filtered = df[df['Min'] > 900].copy()
```

   - Market Value Data: Using Selenium and BeautifulSoup, data was scraped from 'footballtransfers.com' across 22 pages, including player names, ages, clubs, and market values.

```
### BƯỚC 2: Crawl dữ liệu từ web ###
PATH = 'D:/chromedriver/chromedriver-win64/chromedriver-win64/chromedriver.exe'
service = Service(executable_path=PATH)
driver = webdriver.Chrome(service=service)

all_data = []
for page in range(1, 23):
    if page == 1:
        url = 'https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league'
    else:
        url = f'https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league/{page}'
    driver.get(url)
    time.sleep(2)

    soup = BeautifulSoup(driver.page_source, 'html.parser')
    table = soup.find('tbody', {'id': 'player-table-body'})

    rows = table.find_all('tr')
    print(f"✅ Trang {page}: {len(rows)} cầu thủ")

    for row in rows:
        cols = row.find_all('td')
        data = []
        for idx, col in enumerate(cols):
            if idx == 2:  # Cột "Player"
                span = col.find('span')
                name = span.get_text(strip=True) if span else col.get_text(strip=True)
                data.append(name)
            else:
                data.append(col.get_text(strip=True))
        if data:
            all_data.append(data)

driver.quit()
```

- The scraped data was stored in a DataFrame ‘df_values’ with columns: Skill, Rank, Player, Age, Club, MarketValue.

```
### BƯỚC 3: Tạo DataFrame từ dữ liệu crawl ###
df_values = pd.DataFrame(all_data)
df_values.columns = ['Skill', 'Rank', 'Player', 'Age', 'Club', 'MarketValue']
```

2) Data Cleaning and Preprocessing.
   - Player Name Cleaning:
     - Handled duplicated names (e.g., "CaicedoCaicedo" → "Caicedo") using regular expressions.

```
## BƯỚC 4: Làm sạch tên cầu thủ ###
def clean_player_name(name):
    name = str(name)

    # Tên bị lặp (ví dụ: CaicedoCaicedo)
    match = re.match(r'^([A-Za-zÀ-ÿ\.\- ]+?)\1', name)
    if match:
        return match.group(1).strip()
```

- Standardized names by extracting the first two words (if applicable).

```
    # Lấy 2 từ đầu tiên
    words = name.split()
    if len(words) >= 2:
        return f"{words[0]} {words[1]}"
    return name
```

- Data Merging:
  - Applied fuzzy matching (using 'rapidfuzz') to align player names between 'results.csv' and web-scraped data, with a similarity threshold ≥ 85%.
  - Added the 'MarketValue' column to 'df_filtered' (restricted to players with > 900 minutes played).

```
### BƯỚC 5: Fuzzy match tên ###
web_players = df_values['CleanedPlayer'].tolist()
value_dict = dict(zip(df_values['CleanedPlayer'], df_values['MarketValue']))

def get_market_value_fuzzy(player_name):
    match, score, _ = process.extractOne(player_name, web_players, scorer=fuzz.token_sort_ratio)
    if score >=85:
        return value_dict[match]
    return 'N/A'

df_filtered['MarketValue'] = df_filtered['CleanedPlayer'].apply(get_market_value_fuzzy)
```

- Market Value Processing:
  - Converted string values (e.g., "€18.7M" → 18700000) and replaced "N/A" with NaN.

```
# 4. Làm sạch cột MarketValue (chuyển từ chuỗi thành số)
def clean_market_value(value):
    if value == 'N/A' or pd.isna(value):
        return np.nan
    value = value.replace('€', '').strip()
    if 'M' in value:
        return float(value.replace('M', '')) * 1000000
    elif 'K' in value:
        return float(value.replace('K', '')) * 1000
    return float(value)

data['MarketValue'] = data['MarketValue'].apply(clean_market_value)
```

- Removed rows with NaN in 'MarketValue' or key features.

```
# 5. Loại bỏ các hàng có MarketValue là NaN
data = data.dropna(subset=['MarketValue'])
```

- Feature Standardization:
  - Converted ‘Age’ to numeric values (removing date components).

```
# Xử lý cột Age
def clean_age(age):
    if pd.isna(age):
        return np.nan
    age_str = str(age).split('-')[0]
    return float(age_str)

data['Age'] = data['Age'].apply(clean_age)
```

- Removed "%" from percentage columns (Passing_Total_Cmp%, Passing_Medium_Cmp%, Passing_Long_Cmp%).

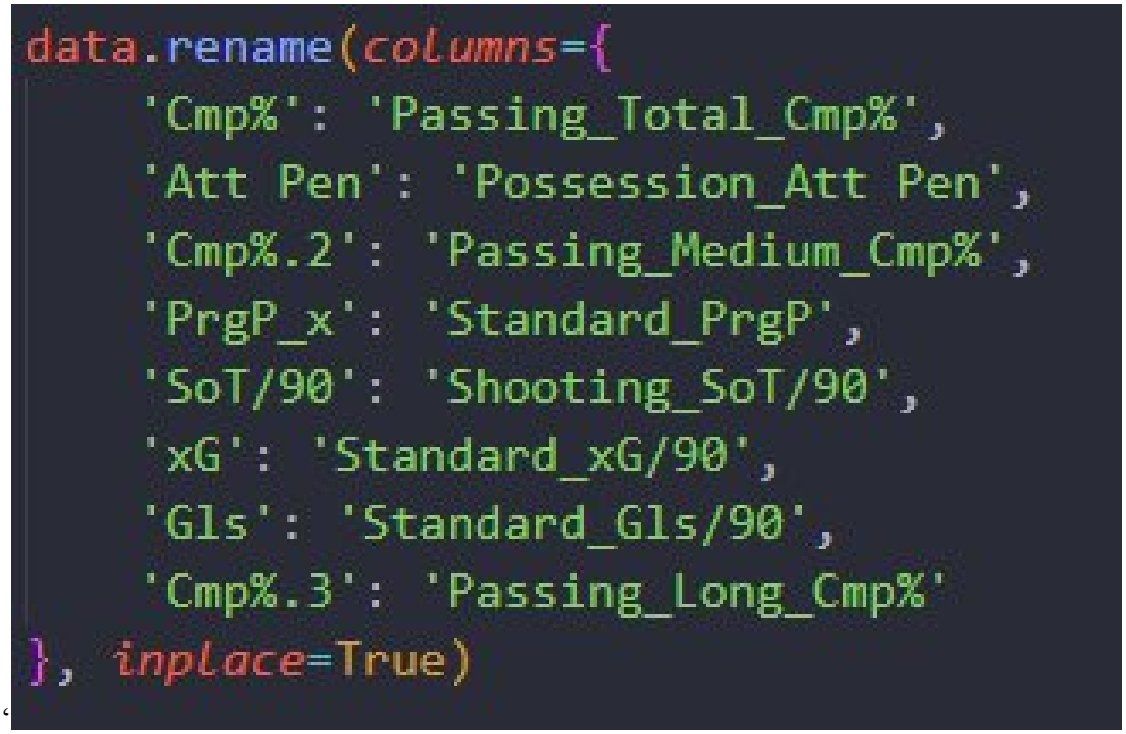
```
# Xử lý các cột phần trăm
def clean_percentage(value):
    if pd.isna(value):
        return np.nan
    return float(str(value).replace('%', ''))

percentage_cols = ['Passing_Total_Cmp%', 'Passing_Medium_Cmp%', 'Passing_Long_Cmp%']
for col in percentage_cols:
    data[col] = data[col].apply(clean_percentage)
```

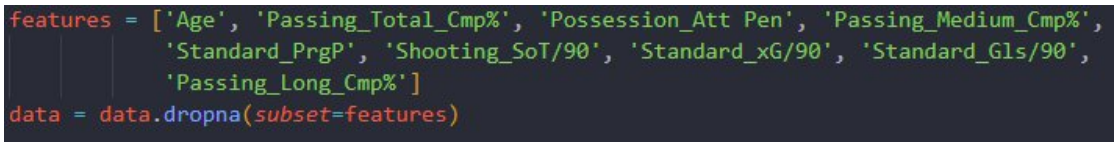
- Converted other metrics (Possession_Att Pen, Standard_PrgP, Shooting_SoT/90, Standard_xG/90, Standard_Gls/90) to numeric format.

```
# Chuyển đổi các cột còn lại thành số
numeric_cols = ['Possession_Att Pen', 'Standard_PrgP', 'Shooting_SoT/90', 'Standard_xG/90', 'Standard_Gls/90']
for col in numeric_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')
```

3) Machine Learning Model Development.

- Selected Features:
    - Age
    - Passing_Total_Cmp%
    - Possession_Att Pen
    - Passing_Medium_Cmp%
    - Standard_PrgP
    - Shooting_SoT/90
    - Standard_xG/90
    - Standard_Gls/90
    - Passing_Long_Cmp%





- Target: ‘MarketValue’

```
# 11. Chọn đặc trưng và mục tiêu
target = 'MarketValue'
X = data[features]
y = data[target]
```

- Data Splitting: The dataset was split into training (80%) and testing (20%) sets with ‘random_state=42’.

```
# 12. Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Models Used:
  - Linear Regression
  - Ridge Regression
  - Lasso Regression
  - Random Forest Regressor
  - XGBoost Regressor

```
# 13. Khởi tạo các mô hình
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'XGBoost': XGBRegressor(random_state=42)
}
```

- Evaluation Metrics:
  - Mean Squared Error (MSE)
  - Mean Absolute Error (MAE)
  - R^2 Score

```
# 14. Lưu trữ kết quả đánh giá
mse_scores = {}
mae_scores = {}
r2_scores = {}
```

4) Feature Importance Analysis.
   - Used the XGBoost model to calculate feature importance scores.
   - Results were saved in ‘feature_importance_1.csv’ and visualized using a horizontal bar chart.

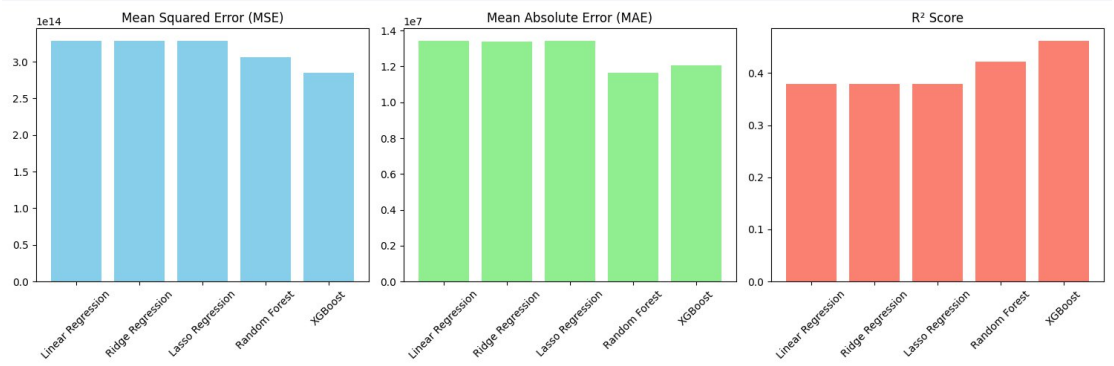
4. Results.

1) Output Data.
   - The ‘results_ex4.csv’ file lists 296 players with columns: ‘stt’, ‘Player’, ‘Min’, ‘MarketValue’. Some players have "N/A" values due to unmatched names or missing web data. Example:

stt	Player	Min	MarketValue
1	Aaron Ramsdale	2250.0	€18.7M
2	Aaron Wan-Bissaka	2704.0	€26.9M
3	Abdoulaye Doucouré	2335.0	€5.8M
4	Adam Armstrong	1248.0	N/A
5	Adam Smith	1319.0	€1.5M
6	Adam Wharton	1258.0	€48.9M
7	Adama Traoré	1523.0	€8M
8	Alejandro Garnacho	1966.0	€60.7M
9	Alex Iwobi	2636.0	N/A
10	Alexander Isak	2411.0	€120.3M

   - A total of 258 players had valid (non-"N/A") market values.

2) Model Performance.
   - Model performance was evaluated using MSE, MAE, and R^2 Score, visualized in bar charts



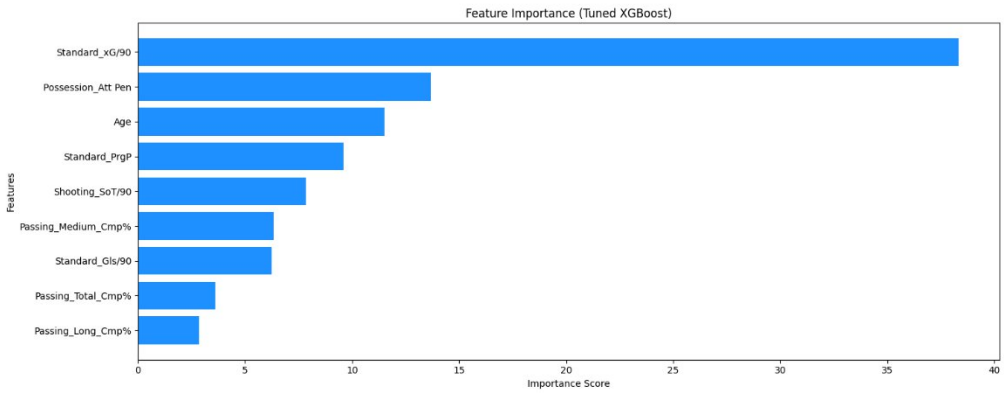
Bar charts comparing model performance (MSE, MAE, R² Score).

- Observations:
  - XGBoost and Random Forest outperformed linear models (Linear, Ridge, Lasso) due to their ability to capture non-linear relationships.
  - XGBoost typically achieved a higher R² Score, indicating better explanation of market value variance.
  - XGBoost also had lower MSE and MAE, suggesting predictions closer to actual values.

3) Feature Importance.
   - The 'feature_importance_1.csv' file provides feature importance scores from the XGBoost model:

Features	Importance Score
Passing_Long_Cmp%	2.8675756
Passing_Total_Cmp%	3.6151004
Standard_Gls/90	6.2397747
Passing_Medium_Cmp%	6.32433
Shooting_SoT/90	7.8329506
Standard_PrgP	9.59917
Age	11.504502
Possession_Att Pen	13.6720915
Standard_xG/90	38.3445

- Analysis:
    - ‘Standard_xG/90’ (expected goals per 90 minutes) was the most important feature (38.3445%), highlighting the critical role of goal-scoring potential in market value.
    - ‘Possession_Att Pen’ (touches in the opponent’s penalty area) ranked second (13.67200915%), reflecting the importance of involvement in dangerous attacking plays.
    - ‘Age’ (11.504502%) had a significant impact, consistent with younger players often commanding higher market values.
    - Passing-related features (Passing_Total_Cmp%, Passing_Medium_Cmp%, Passing_Long_Cmp%) and actual goals (Standard_Gls/90) had lower importance but still contributed to the model.



Horizontal bar chart of feature importance for the XGBoost model.

5. Discussion.

- Strengths:
    - Successfully integrated data from multiple sources (CSV and web) using fuzzy matching to handle name inconsistencies.
    - Compared multiple machine learning models to identify the best performer (XGBoost).
    - Feature importance analysis provided clear insights into factors driving market value.
- Limitations:
    - Some players had "N/A" market values due to unmatched names or missing web data, reducing the dataset size.
    - Linear models (Linear, Ridge, Lasso) performed poorly due to complex non-linear relationships in the data.

- The model may miss external factors (e.g., player branding, contract status, or position).

6. **Conclusion.**

Problem IV successfully collected, cleaned, and analyzed data to predict football players' market values. The XGBoost model demonstrated the best performance, with 'Standard_xG/90', 'Possession_Att Pen', and 'Age' identified as the most influential features. These findings offer valuable insights into the factors driving player market value and showcase the application of machine learning in sports data analysis. The 'results_ex4.csv' and 'feature_importance_1.csv' files serve as evidence of the successful data processing and analysis pipeline.