Bài 1:

1. Khai báo thư viện

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
import pandas as pd
import time
```

- `selenium`: Thư viện Selenium giúp tự động hóa việc điều khiển trình duyệt web. Selenium sẽ giúp chúng ta duyệt trang web, tìm các phần tử trên trang và lấy dữ liệu từ chúng.

- `webdriver`: Đây là module chính trong Selenium, được sử dụng để điều khiển trình duyệt (như Chrome, Firefox).

- `By`: Cung cấp các phương thức để xác định các phần tử trên trang web (ví dụ: qua ID, class, XPath).

- `Service`: Dùng để chỉ định đường dẫn đến ChromeDriver (trình điều khiển của trình duyệt Chrome).

- `Options`: Cho phép cấu hình các tùy chọn cho trình duyệt, ví dụ như chế độ không giao diện người dùng (headless).

- `pandas`: Là thư viện xử lý và phân tích dữ liệu trong Python, rất hữu ích để lưu trữ và xử lý bảng dữ liệu (DataFrame).

- `time`: Thư viện giúp chúng ta tạm dừng chương trình trong một khoảng thời gian (ví dụ: để đợi trang web tải xong).

2. Cấu hình Selenium

```python
chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
Service = Service(executable_path=r"C:\Users\VIETTELSTORE\Downloads\chromedriver-win64\chromedriver-win64\chromedriver.exe")
driver=webdriver.Chrome(options=chrome_options, service=Service)
```

- `chrome_options`: Tạo một đối tượng cấu hình trình duyệt Chrome. Các tùy chọn được thiết lập ở đây:

  - `--headless`: Chạy trình duyệt mà không mở giao diện người dùng (giúp tiết kiệm tài nguyên và tăng tốc độ khi lấy dữ liệu).
  - `--no-sandbox` và `--disable-dev-shm-usage`: Các tùy chọn này giúp tránh lỗi khi chạy trong môi trường máy chủ hoặc các máy tính có hạn chế về tài nguyên.

- `Service`: Chỉ định đường dẫn đến `chromedriver.exe`, công cụ dùng để điều khiển Chrome. Đây là phần bắt buộc khi sử dụng Selenium với Chrome.

- `webdriver.Chrome`: Khởi tạo đối tượng `driver` để điều khiển trình duyệt Chrome theo cấu hình đã thiết lập.

3. URL và Danh mục thống kê

```
url='https://fbref.com/en/comps/9/{}/Premier-League-Stats'
categories = ["stats", "keepers", "shooting","passing", "gca", "defense", "possession", "misc"]
path=["standard", 'keeper', 'shooting','passing', 'gca', 'defense', 'possession', 'misc']
dataframes={}
n=[0,1,2,3,4,5,6,7]
```

- `url`: Đây là URL của trang FBref nơi chứa dữ liệu về cầu thủ Premier League. `{}` sẽ được thay thế bằng từng mục trong danh sách `categories`.

- `categories`: Các danh mục thống kê mà bạn sẽ thu thập dữ liệu từ trang web, bao gồm các chỉ số như: thống kê chung, thủ môn, sút bóng, chuyền bóng, v.v.

- `path`: Là các phần của URL đại diện cho từng loại dữ liệu cụ thể. Mỗi mục trong `categories` tương ứng với một phần tử trong `path`.

- `dataframes`: Tạo một từ điển để lưu trữ dữ liệu từ mỗi danh mục thống kê.

- `n`: Một danh sách chỉ số để lặp qua tất cả các danh mục thống kê (từ 0 đến 7).

4. Cấu hình ánh xạ các cột dữ liệu

```
mapping_config = {
    "stats": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
        "Position": "position",
        "Age": "age",
        "Matches Played": "games",
        "Starts": "games_starts",
        "Minutes": "minutes",
        "Goals": "goals",
        "Assists": "assists",
        "Yellow Cards": "cards_yellow",
        "Red Cards": "cards_red",
        "xG": "xg",
        "xAG": "xg_assist",
        "PrgC": "progressive_carries",
        "PrgP": "progressive_passes",
        "PrgR": "progressive_passes_received",
        "Gls90": "goals_per90",
        "Ast90": "assists_per90",
        "xG90": "xg_per90",
```

```
        "xAG90": "xg_assist_per90"
    },
    "keepers": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
        "Position": "position",
        "Age": "age",
        "Matches Played": "gk_games",
        "Starts": "gk_games_starts",
        "Minutes": "gk_minutes",
        "GA90": "gk_goals_against_per90",
        "Save%": "gk_save_pct",
        "CS%": "gk_clean_sheets_pct",
        "PK Save%": "gk_pens_save_pct"
    },
    "shooting": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
        "Position": "position",
        "Age": "age",
        "SoT%": "shots_on_target_pct",
        "SoT/90": "shots_on_target_per90",
        "G/Sh": "goals_per_shot",
        "Dist": "average_shot_distance"
    },
    "passing": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
        "Position": "position",
        "Age": "age",
        "Cmp": "passes_completed",
        "Cmp%": "passes_pct",
        "TotDist": "passes_total_distance",
        "Cmp% (Short)": "passes_pct_short",
        "Cmp% (Medium)": "passes_pct_medium",
        "Cmp% (Long)": "passes_pct_long",
        "KP": "assisted_shots",
        "Passes 1/3": "passes_into_final_third",
        "PPA": "passes_into_penalty_area",
        "CrsPA": "crosses_into_penalty_area"
    },
    "gca": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
```

```
        "Position": "position",
        "Age": "age",
        "SCA": "sca",
        "SCA90": "sca_per90",
        "GCA": "gca",
        "GCA90": "gca_per90"
    },
    "defense": {
        "Player": "player",
        "Nation": "nationality",
        "Team": "team",
        "Position": "position",
        "Age": "age",
        "Tkl": "tackles",
        "TklW": "tackles_won",
        "Att": "challenges",
        "Challenges Lost": "challenges_lost",
        "Blocks": "blocks",
        "Sh": "blocked_shots",
        "Pass": "blocked_passes",
        "Int": "interceptions"
    },
    "possession": {
        "Player": "player",
        "Nation": "nationality",
              "Team": "team",
              "Position": "position",
              "Age": "age",
              "Touches": "touches",
              "Def Pen": "touches_def_pen_area",
              "Def 3rd": "touches_def_3rd",
              "Mid 3rd": "touches_mid_3rd",
              "Att 3rd": "touches_att_3rd",
              "Att Pen": "touches_att_pen_area",
              "Att (TO)": "take_ons",
              "Succ%": "take_ons_won_pct",
              "Tkld%": "take_ons_tackled_pct",
              "Carries": "carries",
              "ProDist": "carries_progressive_distance",
              "ProgC": "progressive_carries",
              "Carries 1/3": "carries_into_final_third",
              "CPA": "carries_into_penalty_area",
              "Mis": "miscontrols",
              "Dis": "dispossessed",
              "Rec": "passes_received"
          },
          "misc": {
              "Player": "player",
```

```
            "Nation": "nationality",
            "Team": "team",
            "Position": "position",
            "Age": "age",
            "Fls": "fouls",
            "Fld": "fouled",
            "Off": "offsides",
            "Crs": "crosses",
            "Recov": "ball_recoveries",
            "Won": "aerials_won",
            "Aerials Lost": "aerials_lost",
            "Won%": "aerials_won_pct"
        }
    }
```

- `mapping_config`: Đây là cấu hình ánh xạ giữa các tên cột trong dữ liệu lấy từ FBref và các tên cột chuẩn trong DataFrame của bạn. Ví dụ, từ "Goals" trong dữ liệu gốc sẽ được ánh xạ thành "goals" trong DataFrame.

5. Hàm `add`

```python
def add(all_data,col1,col2,col3):
    col_index = all_data.columns.get_loc(col1)
    all_data[col3] = all_data[col1].combine_first(all_data[col2])
    all_data.drop(columns=[col1,col2], inplace=True)
    cols = list(all_data.columns)
    cols.remove(col3)
    cols.insert(col_index,col3)
    return all_data[cols]
```

- Hàm này dùng để hợp nhất hai cột có tên là `col1` và `col2` thành một cột mới tên là `col3`.
- Nếu một trong các giá trị ở `col1` là `NaN`, giá trị từ `col2` sẽ thay thế.
- Sau đó, hàm sẽ xóa đi hai cột ban đầu và giữ cột mới tại đúng vị trí.

6. Hàm `restructure_row`

```python
def restructure_row(row_data, mapping):
    new_data = {}
    for field, header in mapping.items():
        new_data[field] = row_data.get(header, "N/A")
    return new_data
```

- Hàm này sẽ nhận vào một dòng dữ liệu và ánh xạ các giá trị từ `row_data` theo cấu hình trong `mapping`. Nếu một giá trị không có, nó sẽ gán "N/A".

- Kết quả trả về là một từ điển với các trường đã được chuẩn hóa.

7. Thu thập và xử lý dữ liệu

```
for i in n:
    player = []
    driver.get(url.format(categories[i]))
    time.sleep(3)
    header_elements = driver.find_elements(By.XPATH, '//table[@id="stats_{}"]//thead//tr[2]//th'.format(path[i]))
    headers = [h.get_attribute('data-stat') for h in header_elements]
    row_elements = driver.find_elements(By.XPATH, '//table[@id="stats_{}"]//tbody//tr'.format(path[i]))
    headers.pop(0)
    for row in row_elements:
        cell_elements = row.find_elements(By.TAG_NAME, 'td')
        row_data = {header: cell.text.strip() for header, cell in zip(headers, cell_elements)}
        restructured = restructure_row(row_data, mapping_config[categories[i]])
        player.append(restructured)
    df = pd.DataFrame(player)
    dataframes[categories[i]] = df
    del df
all_data=dataframes[categories[0]]
for i in range(1,8):
    df = dataframes[categories[i]]
    df = df.groupby("Player").first().reset_index()   # <- dòng thêm vào
    all_data = pd.merge(all_data, df, on=["Player","Nation","Team","Position","Age"], how='outer')
    del df
```

- Mã này lặp qua từng danh mục trong `categories`, lấy dữ liệu từ trang web bằng cách sử dụng `driver.get` để tải trang.
- Sau khi trang web tải xong, sử dụng `driver.find_elements` để lấy các phần tử cần thiết từ bảng (ví dụ: các tiêu đề cột và các dòng dữ liệu).
- Các dữ liệu thu thập được sẽ được xử lý và chuẩn hóa thông qua hàm `restructure_row`, rồi lưu vào `DataFrame`.

8. Kết hợp dữ liệu từ các danh mục thống kê

```
180  for i in range(1,8):
181      df = dataframes[categories[i]]
182      df = df.groupby("Player").first().reset_index()   # <- dòng thêm vào
183      all_data = pd.merge(all_data, df, on=["Player","Nation","Team","Position","Age"], how='outer')
184      del df
```

- Tạo `all_data` từ dữ liệu của danh mục đầu tiên (`stats`).

- Sau đó, lặp qua các danh mục còn lại, nhóm dữ liệu theo "Player" (cầu thủ) và kết hợp với `all_data` bằng phương thức `merge`.

9. Xử lý các cột và lọc dữ liệu

```
186  all_data= add(all_data, 'Minutes_x', 'Minutes_y', 'Minutes')
187  all_data= add(all_data, 'Matches Played_x', 'Matches Played_y', 'Matches Played')
188  all_data= add(all_data, 'Starts_x', 'Starts_y', 'Starts')
189  all_data['Minutes'] = pd.to_numeric(all_data['Minutes'].astype(str).str.replace(",", ""), errors='coerce')
190  all_data=all_data[all_data['Minutes'] > 90]
```

- Dữ liệu từ các cột "Minutes" và "Matches Played" được hợp nhất bằng hàm `add`.

- Cột "Minutes" được chuyển sang kiểu số thực (`float`), và chỉ giữ lại các cầu thủ có trên 90 phút thi đấu.

10. Tiền xử lý dữ liệu và lưu vào file CSV

```
all_data['First Name'] = all_data['Player'].str.split().str[0]
all_data['Last Name']  = all_data['Player'].str.split().str[-1]
all_data = all_data.sort_values(by="First Name")
all_data = all_data.drop(columns=['First Name', 'Last Name'])
all_data = all_data.fillna("N/a")
all_data = all_data.replace("", "N/a")
all_data.to_csv('results.csv', index=False)
```

- Tách tên cầu thủ thành "First Name" và "Last Name", sau đó sắp xếp dữ liệu theo "First Name".
- Thay thế các giá trị thiếu hoặc rỗng bằng "N/a".
- Lưu dữ liệu vào file `results.csv`.

11. Thoát khỏi trình duyệt

```
driver.quit()
```

- Sau khi hoàn tất thu thập và xử lý dữ liệu, đóng trình duyệt để giải phóng tài nguyên.

Bài 2: Phân tích thống kê dữ liệu các chỉ số cầu thủ

1. Đọc dữ liệu và xử lý chỉ số cần phân tích

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

# ======== 1. Đọc dữ liệu và xử lý chỉ số cần phân tích ========
df = pd.read_csv("results.csv")
columns = ['Age', 'Matches Played', 'Starts', 'Minutes', 'Goals', 'Assists', 'Yellow Cards', 'Red Cards',
           'xG', 'xAG', 'PrgC', 'PrgP', 'PrgR', 'Gls90', 'Ast90', 'xG90', 'xAG90', 'GA90', 'Save%',
           'CS%', 'PK Save%', 'SoT%', 'SoT/90', 'G/Sh', 'Dist', 'Cmp', 'Cmp%', 'TotDist', 'Cmp% (Short)',
           'Cmp% (Medium)', 'Cmp% (Long)', 'KP', 'Passes 1/3', 'PPA', 'CrsPA', 'SCA', 'SCA90', 'GCA', 'GCA90',
           'Tkl', 'TklW', 'Att', 'Challenges Lost', 'Blocks', 'Sh', 'Pass', 'Int', 'Touches', 'Def Pen',
           'Def 3rd', 'Mid 3rd', 'Att 3rd', 'Att Pen', 'Att (TO)', 'Succ%', 'Tkld%', 'Carries', 'ProDist',
           'ProgC', 'Carries 1/3', 'CPA', 'Mis', 'Dis', 'Rec', 'Fls', 'Fld', 'Off', 'Crs', 'Recov', 'Won',
           'Aerials Lost', 'Won%']
teams = df['Team'].unique()

```

- Đoạn mã này sử dụng thư viện `pandas` để đọc dữ liệu từ file CSV (`results.csv`) vào một DataFrame.
- Dữ liệu này bao gồm thông tin về các cầu thủ bóng đá, với nhiều chỉ số thống kê như "Goals", "Assists", "Minutes", "xG", và các chỉ số khác.
- Mảng `columns` chứa danh sách các chỉ số mà bạn muốn phân tích.

- Biến `teams` chứa danh sách các đội bóng tham gia trong dữ liệu, giúp phân nhóm các cầu thủ theo đội sau này.

2. Chuyển đổi giá trị "Age" từ dạng sang số năm

```python
# ======== 2. Chuyển đổi giá trị "Age" sang số năm ========
def convert_age_to_years(age_str):
    try:
        # Tách tuổi và số ngày
        age, days = age_str.split('-')
        age = int(age)
        days = int(days)
        # Tính tuổi tính theo năm, cộng thêm phần năm từ số ngày
        return age + (days / 365.25)
    except Exception as e:
        print(f"Lỗi khi chuyển đổi tuổi {age_str}: {e}")
        return 0  # Trả về 0 nếu không thể chuyển đổi

# Áp dụng hàm chuyển đổi cho cột 'Age'
df['Age'] = df['Age'].apply(convert_age_to_years)
```

- Cột `Age` chứa giá trị tuổi của các cầu thủ dưới dạng `XX-YYY`, trong đó `XX` là số tuổi và `YYY` là số ngày trong năm tính từ ngày sinh.

- Đoạn mã này định nghĩa một hàm `convert_age_to_years` để tách phần tuổi và phần ngày, sau đó tính toán tuổi chính xác theo năm.

- Việc chia cho 365.25 giúp tính đúng số ngày trong một năm tính cả năm nhuận.

- Hàm này được áp dụng cho toàn bộ cột `Age` của DataFrame, chuyển đổi tất cả các giá trị thành kiểu dữ liệu số (float), giúp dễ dàng xử lý và phân tích dữ liệu.

3. **Chuyển đổi các cột còn lại về kiểu số**

```python
# ======== 3. Chuyển đổi các cột còn lại về kiểu số ========
for col in columns:
    df[col] = pd.to_numeric(df[col].astype(str).str.replace(",", "").replace("N/a", "0"), errors='coerce')
```

- Sau khi xử lý cột `Age`, đoạn mã này tiếp tục chuyển đổi các cột còn lại trong `columns` về kiểu dữ liệu số.

- Đối với mỗi cột, hàm `apply` sẽ loại bỏ dấu phẩy (`,`), chuyển đổi giá trị thành số và thay thế các giá trị "N/a" bằng 0.

- `errors='coerce'` đảm bảo rằng những giá trị không hợp lệ sẽ bị thay thế bằng `NaN`, tránh lỗi trong quá trình tính toán sau này.

4. **Top 3 cao nhất và thấp nhất mỗi chỉ số**

```python
# ======== 4. Top 3 cao nhất và thấp nhất mỗi chỉ số ========
with open("top_3.txt", "w", encoding="utf-8") as f:
    for col in columns:
        if df[col].dropna().empty:
            continue
        top_3 = df[['Player', col]].sort_values(by=col, ascending=False).head(3)
        bottom_3 = df[['Player', col]].sort_values(by=col, ascending=True).head(3)

        f.write(f"=== Top 3 for {col} ===\n")
        for _, row in top_3.iterrows():
            f.write(f"{row['Player']}: {row[col]}\n")
        f.write(f"\n=== Bottom 3 for {col} ===\n")
        for _, row in bottom_3.iterrows():
            f.write(f"{row['Player']}: {row[col]}\n")
        f.write("\n" + "="*50 + "\n\n")

print("✅ Đã tạo file top_3.txt")

```

- Đoạn mã này tìm ra các cầu thủ có chỉ số cao nhất và thấp nhất trong mỗi cột trong danh sách `columns`.

- Đối với mỗi cột, các cầu thủ được sắp xếp theo giá trị của cột đó và lấy 3 cầu thủ có giá trị cao nhất và thấp nhất.

- Kết quả được ghi vào file `top_3.txt` với tên cầu thủ và giá trị tương ứng.

- Đây là cách để so sánh các cầu thủ trong các chỉ số quan trọng, giúp bạn tìm ra những cầu thủ nổi bật hoặc thiếu sót trong từng chỉ số.

5. Tính median, mean, std theo đội bóng

```python
# ======== 5. Median/Mean/Std theo đội bóng ========
results = []

# Kiểm tra giá trị hợp lệ trong mỗi đội
for team in teams:
    row = {'Team': team}
    for col in columns:
        team_df = df[df['Team'] == team]

        # Kiểm tra xem dữ liệu trong cột có hợp lệ
        valid_values = team_df[col].dropna()
        if not valid_values.empty:
            # Tính các chỉ số nếu có giá trị hợp lệ
            row[f'Median of {col}'] = valid_values.median()
            row[f'Mean of {col}'] = valid_values.mean()
            row[f'Std of {col}'] = valid_values.std()
        else:
            # Nếu không có giá trị hợp lệ, gán giá trị mặc định
            row[f'Median of {col}'] = 0.0
            row[f'Mean of {col}'] = 0.0
            row[f'Std of {col}'] = 0.0

    results.append(row)

# Lưu kết quả vào file CSV
pd.DataFrame(results).round(3).fillna(0.0).to_csv("results2.csv", index=False)
print("✅ Đã tạo file results2.csv")
```

- Đoạn mã này tính toán các chỉ số thống kê cơ bản (median, mean, và standard deviation) cho từng chỉ số trong `columns`, nhưng phân nhóm theo đội bóng.

- Kết quả được lưu vào một file `team_stats.csv`, giúp bạn phân tích các đội bóng dựa trên các chỉ số của các cầu thủ trong đội đó.

- Các phép tính thống kê này sẽ giúp bạn so sánh các đội bóng về các chỉ số hiệu suất và ra quyết định trong việc phân tích.

6. Tìm đội có chỉ số trung bình cao nhất cho từng tiêu chí

```python
# ======== 6. Tìm đội có chỉ số cao nhất mỗi tiêu chí ========
best_teams = {}
for col in columns:
    team_means = df.groupby('Team')[col].mean()
    best_team = team_means.idxmax()
    best_teams[col] = best_team

with open("best_teams.txt", "w") as f:
    for stat, team in best_teams.items():
        f.write(f'Top team for {stat}: {team}\n')

print("✅ Đã tạo file best_teams.txt")
```

- Đoạn mã này xác định đội bóng nào có **giá trị trung bình cao nhất** cho từng chỉ số trong danh sách `columns`.

- Dữ liệu được nhóm theo `Team`, sau đó tính trung bình (`mean`) cho từng chỉ số.

- Đội có giá trị trung bình cao nhất (`idxmax()`) được ghi nhận là đội "tốt nhất" trong chỉ số đó.

- Kết quả được lưu vào file `best_teams.txt`, giúp bạn dễ dàng xác định đội nổi bật nhất theo từng tiêu chí.

7. Vẽ biểu đồ histogram cho 3 chỉ số tấn công và phòng thủ toàn giải và từng đội

```python
# ======== 7. Vẽ biểu đồ histogram từng chỉ số toàn giải và từng đội ========
df = pd.read_csv("results.csv")
columns = ['PrgC', 'PrgR', 'PrgP', 'Save%', 'PK Save%', 'CS%']
teams = df['Team'].unique()
os.makedirs("histograms", exist_ok=True)

for col in columns:
    # Toàn giải
    plt.figure(figsize=(10,6))
    plt.hist(df[col].dropna(), bins=20, alpha=0.7, color='blue')
    plt.title(f'Distribution of {col} (All Players)')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.savefig(f'histograms/{col}_all.png')
    plt.close()

    # Từng đội
    for team in teams:
        team_df = df[df['Team'] == team]
        plt.figure(figsize=(10,6))
        plt.hist(team_df[col].dropna(), bins=20, alpha=0.7, color='green')
        plt.title(f'Distribution of {col} ({team})')
        plt.xlabel(col)
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.savefig(f'histograms/{col}_{team}.png')
        plt.close()

print("✅ Đã tạo biểu đồ histogram trong thư mục histograms/")
```

- Biểu đồ histogram giúp trực quan hóa phân bố giá trị của từng chỉ số, từ đó hiểu được độ trải đều hay chênh lệch.

- Đầu tiên, tạo thư mục `histograms/` để lưu các hình ảnh biểu đồ.

- Với mỗi chỉ số trong `columns`, vẽ:

  - Một biểu đồ histogram cho toàn bộ giải đấu (tất cả cầu thủ).
  - Một biểu đồ riêng cho từng đội bóng, giúp so sánh phân bố nội bộ.

- Điều này hỗ trợ phát hiện các đội có xu hướng vượt trội hoặc tụt hậu ở một chỉ số cụ thể.

Bài 3: Phân cụm cầu thủ bằng K-means và giảm chiều PCA

1. **Đọc dữ liệu và xử lý giá trị số**

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Đọc dữ liệu
df = pd.read_csv('results.csv')

# Chuyển 'N/a' thành NaN và ép kiểu numeric cho các cột số
df.replace('N/a', np.nan, inplace=True)

```

- Đọc dữ liệu từ file `results.csv`
- Thay thế các giá trị `"N/a"` bằng `NaN` để xử lý dữ liệu thiếu

2. **Xác định và chuyển đổi các cột số**

```python
# Xác định các cột số
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

# Nếu chưa có numeric cols – ép kiểu thử các cột còn lại
if not numeric_cols:
    for col in df.columns:
        try:
            df[col] = pd.to_numeric(df[col])
        except:
            continue
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()

```

- Tự động xác định các cột dạng số (`float`/`int`)
- Nếu tất cả cột đang ở dạng chuỗi (`object`), cố gắng chuyển đổi sang kiểu số

3. **Xử lý thiếu dữ liệu và chuẩn hóa**

```python
# Loại bỏ các hàng bị thiếu dữ liệu
df.dropna(subset=numeric_cols, inplace=True)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[numeric_cols])
```

- Bỏ các cầu thủ thiếu dữ liệu ở bất kỳ cột chỉ số nào
- Chuẩn hóa dữ liệu về cùng thang đo (mean=0, std=1) để phân cụm hiệu quả

4. Elbow Method tìm số lượng cụm tối ưu (k)

```python
# Tìm số cluster tối ưu bằng Elbow Method
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Vẽ biểu đồ Elbow
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid()
plt.show()
```

- Chạy thuật toán K-means với các giá trị `k = 1` đến `10`
- Ghi lại độ biến thiên (`inertia`) của từng mô hình
- Vẽ biểu đồ Elbow để chọn số lượng cụm hợp lý (nơi "gập khúc")

5. Tiến hành phân cụm với k cụ thể (ví dụ: k=3)

```python
# Chọn số cluster – ví dụ k=3 từ Elbow Method
k = 3
kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Thêm cột cluster vào dataframe
df['Cluster'] = clusters
```

- Dùng K-means để phân chia dữ liệu thành `k=3` nhóm

- Gán kết quả vào cột `Cluster` trong dataframe để tiện theo dõi nhóm của từng cầu thủ

6. Giảm chiều dữ liệu với PCA xuống 2 chiều

```
58
59  # Giảm chiều dữ liệu xuống 2D với PCA
60  pca = PCA(n_components=2)
61  X_pca = pca.fit_transform(X_scaled)
62
63  # Thêm PCA vào dataframe
64  df['PCA1'] = X_pca[:, 0]
65  df['PCA2'] = X_pca[:, 1]
66
```

- Dùng PCA để chuyển dữ liệu đa chiều thành 2 chiều (`PCA1`, `PCA2`)

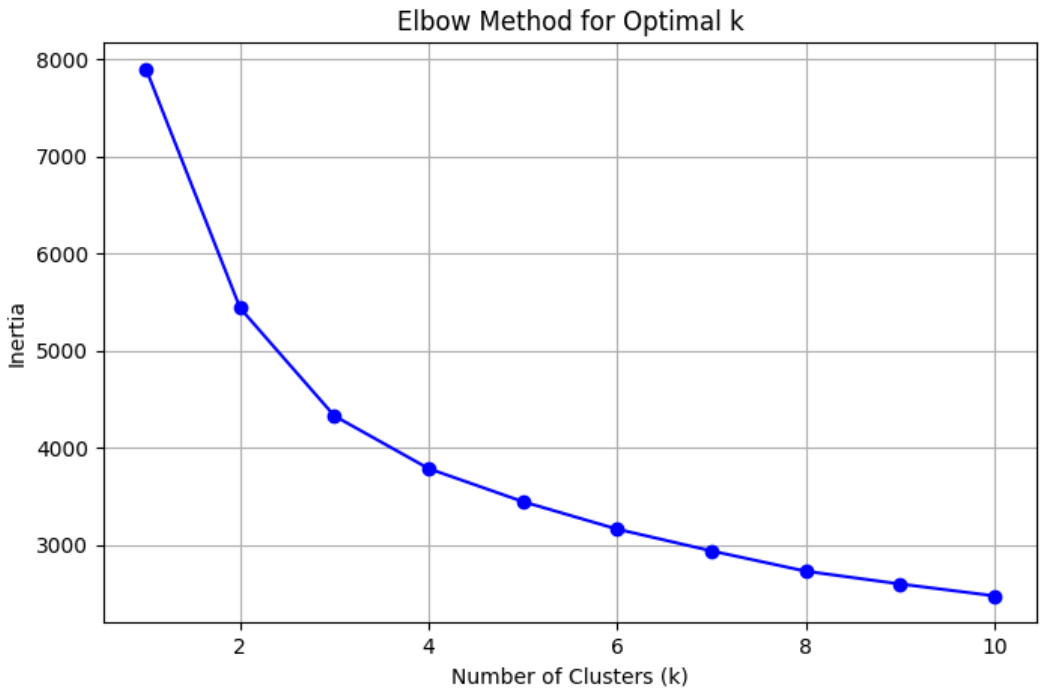
- Giúp trực quan hóa dữ liệu phân cụm dễ hiểu hơn

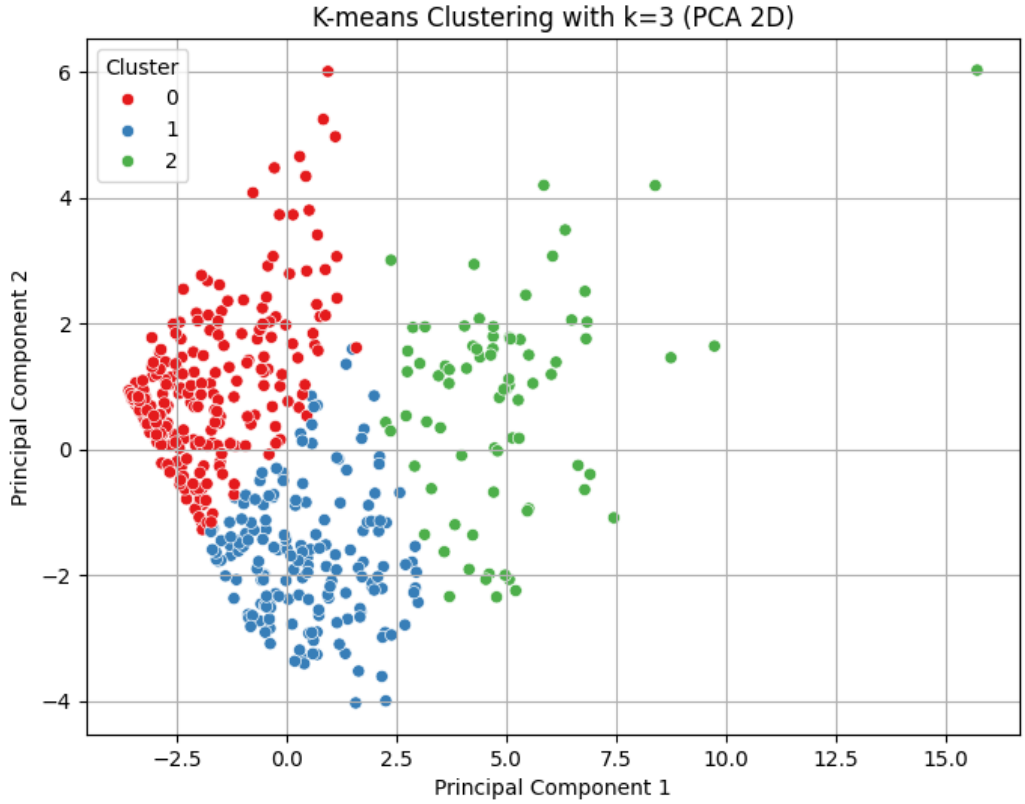
7. Vẽ biểu đồ phân cụm cầu thủ theo PCA

```
67  # Vẽ scatter plot các nhóm
68  plt.figure(figsize=(8, 6))
69  sns.scatterplot(data=df, x='PCA1', y='PCA2', hue='Cluster', palette='Set1')
70  plt.title(f'K-means Clustering with k={k} (PCA 2D)')
71  plt.xlabel('Principal Component 1')
72  plt.ylabel('Principal Component 2')
73  plt.legend(title='Cluster')
74  plt.grid(True)
75  plt.show()
76
```

- Vẽ biểu đồ phân tán (`scatter plot`) thể hiện các cụm cầu thủ trên mặt phẳng 2D

- Mỗi màu đại diện cho 1 cụm → dễ so sánh cách các cầu thủ được nhóm lại theo phong cách, vai trò, hoặc năng lực





Bài 4 – Phần 1: Thu thập giá trị chuyển nhượng cầu thủ

1. Import thư viện cần thiết

```python
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
import time
```

- **`pandas`**: xử lý dữ liệu
- **`selenium`**: tự động hóa việc truy cập và lấy dữ liệu từ website
- **`time.sleep`**: tạm dừng để website kịp tải nội dung

2. Đọc dữ liệu đầu vào và lọc cầu thủ hợp lệ

```python
# --- Phần 1: Thu thập giá trị chuyển nhượng cầu thủ ---

# Đọc dữ liệu từ results.csv
df = pd.read_csv("results.csv")
df.columns = df.columns.str.lower()  # Đảm bảo cột được viết thường

# Lọc cầu thủ có thời gian thi đấu > 900 phút
df_filtered = df[df["minutes"] > 900]
players_needed = set(df_filtered["player"].str.strip())

```

- Đọc dữ liệu từ file `results.csv` đã có từ các phần trước
- Lọc các cầu thủ có số phút thi đấu > 900 phút
- Tạo danh sách tên cầu thủ cần thu thập giá trị chuyển nhượng

3. Cấu hình Selenium với Chrome headless

```python
# Cấu hình Selenium
chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
service = Service(executable_path=r"C:\Users\VIETTELSTORE\Downloads\chromedriver-win64\chromedriver-win64\chromedriver.exe")
driver = webdriver.Chrome(options=chrome_options, service=service)

```

- Dùng Chrome ở chế độ `headless` để không cần mở trình duyệt
- Chỉ định đường dẫn đến ChromeDriver để Selenium điều khiển trình duyệt

4. Truy cập và thu thập dữ liệu từ nhiều trang

```python
# URL của các trang cần truy cập
base_url = "https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league/"

# Danh sách để lưu kết quả
transfer_data = []

# Lấy dữ liệu từ các trang 1 đến 22
for page in range(1, 23):
    driver.get(f"{base_url}{page}")
    time.sleep(3)

    # Lấy tất cả tên cầu thủ và giá trị chuyển nhượng
    players = driver.find_elements(By.XPATH, "//td[contains(@class, 'td-player')]//a")
    values = driver.find_elements(By.XPATH, "//td[contains(@class, 'text-center')]//span[contains(@class, 'player-tag')]")

    for player, value in zip(players, values):
        player_name = player.text.strip()
        transfer_value = value.text.strip()

        # Kiểm tra nếu cầu thủ có trong danh sách đã lọc
        if player_name.lower() in [p.lower() for p in players_needed]:
            transfer_data.append({
                "Player": player_name,
                "Value": transfer_value
            })

```

- Duyệt qua 22 trang danh sách cầu thủ Premier League trên FootballTransfers
- Mỗi trang:
  - Lấy danh sách tên cầu thủ (`td-player`)
  - Lấy giá trị chuyển nhượng (`player-tag`)
- Chỉ giữ lại cầu thủ có tên trong danh sách lọc `players_needed`
- Thêm dữ liệu vào danh sách `transfer_data`

5. Lưu dữ liệu vào file CSV và đóng trình duyệt

```python
# Lưu kết quả vào file CSV
transfer_df = pd.DataFrame(transfer_data)
transfer_df.to_csv("transfer_values.csv", index=False)

# Đóng trình duyệt
driver.quit()

print("Lấy dữ liệu thành công và lưu vào file 'transfer_values.csv'.")
```

Câu 4 – Phần 2 - Xây dựng mô hình định giá cầu thủ:

1. Cài đặt thư viện và môi trường

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import logging
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.stats import spearmanr, randint, uniform
from math import sqrt
import xgboost as xgb

```

- Gọi các thư viện cần thiết để xử lý dữ liệu (pandas, numpy), vẽ biểu đồ (matplotlib, seaborn), và kiểm soát môi trường thực thi (warnings, logging, os).

- Import các thành phần học máy từ `scikit-learn` để:

  - Tiền xử lý dữ liệu.
  - Xây dựng pipeline học máy.
  - Tối ưu mô hình.
  - Đánh giá độ chính xác dự đoán.
  - Sử dụng mô hình hồi quy XGBoost.

```python
# Cài đặt môi trường và hiển thị
warnings.filterwarnings("ignore")
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
logging.getLogger('sklearn').setLevel(logging.CRITICAL)
plt.style.use('ggplot')

```

Giảm bớt cảnh báo không cần thiết, tối ưu hiển thị và chọn phong cách vẽ biểu đồ.

2. Load dữ liệu

```python
# --- Load dữ liệu ---
def load_data():
    features = pd.read_csv("results.csv")
    values = pd.read_csv("transfer_values.csv")

    # Chuẩn hóa tên cầu thủ
    for df in [features, values]:
        df["Player"] = df["Player"].str.strip().str.lower()

    # Merge theo "Player"
    merged = pd.merge(features, values, on="Player", how="inner", suffixes=('', '_drop'))
    merged = merged.filter(regex='^(?!.*_drop)')  # Bỏ các cột bị trùng

    return merged

```

Hàm này đọc dữ liệu từ hai file CSV, chuẩn hóa tên cầu thủ để tránh lỗi khi merge. Kết quả là một DataFrame đã ghép nối đầy đủ thông tin.

3. Tiền xử lý dữ liệu

```python
def preprocess_data(df):
    # Chuyển đổi giá trị từ dạng "€10m" -> 10_000_000
    def convert_value(value):
        try:
            value = str(value).lower().replace("€", "").replace(",", "").strip()
            if 'm' in value:
                return float(value.replace('m', '')) * 1e6
            if 'k' in value:
                return float(value.replace('k', '')) * 1e3
            return float(value)
        except:
            return np.nan

    df["Value"] = df["Value"].apply(convert_value)
    df = df[df["Value"] > 1000]  # Loại bỏ cầu thủ không có giá trị thực

    # Chuyển các cột bắt buộc sang số nếu chưa đúng kiểu
    numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
    for col in ['Goals', 'Assists', 'Age']:
        if col in df.columns and col not in numeric_cols:
            df[col] = pd.to_numeric(df[col], errors='coerce')
            numeric_cols.append(col)

    # Thêm đặc trưng mới
    df['Goals_per_Age'] = df['Goals'] / df['Age']
    df['Assists_per_Age'] = df['Assists'] / df['Age']
    df['Log_Transfer_Value'] = np.log1p(df['Value'])

    # Xử lý missing value
    df[['Goals_per_Age', 'Assists_per_Age']] = df[['Goals_per_Age', 'Assists_per_Age']].fillna(0)
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())

    return df

```

Hàm xử lý dữ liệu như chuyển `"€10m"` thành `10000000`, lọc giá trị bất hợp lệ, chuẩn hóa số liệu, thêm đặc trưng chia theo độ tuổi và logarit hóa giá trị chuyển nhượng để giúp mô hình học tốt hơn.

4. Xây dựng pipeline mô hình

```python
# --- Xây dựng pipeline model ---
def build_model(numeric_features):
    preprocessor = ColumnTransformer(transformers=[
        ('num', StandardScaler(), numeric_features)
    ])

    model = Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', xgb.XGBRegressor(objective='reg:squarederror', random_state=42, n_jobs=-1, verbosity=0))
    ])

    return model

```

Pipeline gồm hai bước: (1) Chuẩn hóa dữ liệu số bằng `StandardScaler`, (2) Huấn luyện mô hình hồi quy `XGBoost` để dự đoán log giá trị chuyển nhượng.

5. Đánh giá mô hình

```python
# --- Đánh giá mô hình ---
def evaluate_model(model, X, y, log_target=True):
    pred = model.predict(X)
    if log_target:
        y_true = np.expm1(y)
        pred = np.expm1(pred)
    else:
        y_true = y

    rmse = sqrt(mean_squared_error(y_true, pred))
    mae = mean_absolute_error(y_true, pred)
    r2 = r2_score(y_true, pred)
    spear_corr, _ = spearmanr(y_true, pred)

    print(f"  R2 Score: {r2:.4f}")
    print(f"  RMSE: {rmse / 1e6:.4f} triệu €")
    print(f"  MAE: {mae / 1e6:.4f} triệu €")
    print(f"  Spearman Corr: {spear_corr:.4f}")

```

Hàm đánh giá mô hình bằng các chỉ số:

- R^2: mức độ giải thích biến mục tiêu.
- RMSE & MAE: sai số dự đoán.
- Hệ số tương quan Spearman: độ liên hệ giữa giá trị thật và dự đoán.

6. Huấn luyện và tối ưu mô hình

```python
if __name__ == "__main__":
    df = load_data()
    df = preprocess_data(df)

    # Tách biến đầu vào và đầu ra
    X = df.drop(columns=['Value', 'Log_Transfer_Value', 'Player'])
    y = df['Log_Transfer_Value']
    numeric_features = X.select_dtypes(include=np.number).columns.tolist()

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

    model = build_model(numeric_features)

    # Random search để tối ưu tham số
    param_dist = {
        'regressor__n_estimators': randint(100, 400),
        'regressor__learning_rate': uniform(0.05, 0.2),
        'regressor__max_depth': randint(4, 8),
        'regressor__subsample': uniform(0.7, 0.3),
        'regressor__colsample_bytree': uniform(0.7, 0.3)
    }

    search = RandomizedSearchCV(model, param_dist, n_iter=15, cv=3,
                                scoring='neg_root_mean_squared_error',
                                random_state=42, n_jobs=-1)
    search.fit(X_train, y_train)
    best_model = search.best_estimator_

    print("\nĐánh giá trên tập huấn luyện:")
    evaluate_model(best_model, X_train, y_train)

    print("\nĐánh giá trên tập kiểm tra:")
    evaluate_model(best_model, X_test, y_test)
```

- Tách tập dữ liệu thành huấn luyện và kiểm tra (80/20).
- Áp dụng **RandomizedSearchCV** để chọn tham số tốt nhất cho mô hình `XGBoost`.
- In kết quả đánh giá cả trên tập huấn luyện và kiểm tra.