HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BỘ MÔN LẬP TRÌNH PYTHON

BÁO CÁO BÀI TẬP LỚN PYTHON

Giảng viên : Kim Ngọc Bách

Sinh viên : Nguyễn Duy Khánh

Mã sinh viên: B22DCKH067

Hà Nội, ngày 3 tháng 11 năm 2024

CÂU 1: Thu thập dữ liệu thống kê [*] của tất cả các cầu thủ có số phút thi đấu nhiều hơn 90 phút tại giải bóng đá ngoại hạng Anh mùa 2023-2024.

1. Giới thiệu

- Mã nguồn được cung cấp nhằm mục đích cào dữ liệu (web scraping) về các cầu thủ bóng đá thi đấu trong giải Ngoại hạng Anh (Premier League) mùa giải 2023-2024 từ trang web FBref.com. Dữ liệu thu thập bao gồm thông tin cá nhân, thống kê thi đấu và các chỉ số chuyên môn khác nhau của cầu thủ. Kết quả cuối cùng được lưu trữ dưới dạng tệp CSV để phục vụ cho việc phân tích và nghiên cứu sau này.

2. Thư viện và công cụ sử dụng

- requests: Thư viện dùng để gửi yêu cầu HTTP và nhận nội dung từ trang web.
- BeautifulSoup: Thư viện dùng để phân tích cú pháp HTML và trích xuất thông tin cần thiết.
- pandas: Thư viện hỗ trợ thao tác và quản lý dữ liệu dưới dạng DataFrame.
- time và sys: Thư viện hỗ trợ thao tác với thời gian và hệ thống.

3. Cấu trúc và chức năng của mã nguồn

3.1. Hàm process_footballer_data

```python
def process_footballer_data(player_row, team_name):
    # Hàm lấy giá trị từ một cột nhất định
    def get_stat(stat):
        cell = player_row.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    # Lấy thông tin cầu thủ từ các cột cần thiết
    player_data = {
        "Player Name": player_row.find('th', {'data-stat': 'player'}).get_text(strip=True),
        "Nationality": player_row.find('td', {'data-stat': 'nationality'}).find('a')['href'].split('/')[-1].replace('-Foo
        "Team": team_name,
        "Position": get_stat('position'),
        "Age": get_stat('age'),
        "Games": get_stat('games'),
        "Games Starts": get_stat('games_starts'),
        "Minutes": get_stat('minutes'),
        "Goals (Pens)": get_stat('goals_pens'),
        "Penalties Made": get_stat('pens_made'),
        "Assists": get_stat('assists'),
        "Yellow Cards": get_stat('cards_yellow'),
        "Red Cards": get_stat('cards_red'),
        "xG": get_stat('xg'),
        "npxG": get_stat('npxg'),
        "xAG": get_stat('xg_assist'),
        "Progressive Carries": get_stat('progressive_carries'),
        "Progressive Passes": get_stat('progressive_passes'),
        "Progressive Passes Received": get_stat('progressive_passes_received'),
        "Goals per 90": get_stat('goals_per90'),
        "Assists per 90": get_stat('assists_per90'),
        "Goals + Assists per 90": get_stat('goals_assists_per90'),
        "Goals (Pens) per 90": get_stat('goals_pens_per90'),
        "Goals + Assists (Pens) per 90": get_stat('goals_assists_pens_per90'),
        "xG per 90": get_stat('xg_per90'),
        "xAG per 90": get_stat('xg_assist_per90'),
        "xG + xAG per 90": get_stat('xg_xg_assist_per90'),
        "npxG per 90": get_stat('npxg_per90'),
        "npxG + xAG per 90": get_stat('npxg_xg_assist_per90')
    }

    return list(player_data.values())
```

- Mục đích: Xử lý và trích xuất thông tin cơ bản của cầu thủ từ bảng dữ liệu.

- Tham số:
- player_row: Dòng HTML chứa thông tin của một cầu thủ.
- team_name: Tên đội bóng của cầu thủ.
- Chức năng:
- Sử dụng hàm get_stat để lấy dữ liệu từ các cột cụ thể.
- Trả về danh sách chứa thông tin cá nhân và các chỉ số cơ bản của cầu thủ.

3.2. Các hàm xử lý dữ liệu khác

- process_goalkeeper_data: Xử lý dữ liệu liên quan đến thủ môn.

```
# Hàm xử lý dữ liệu thủ môn
def process_goalkeeper_data(player):
    # Hàm phụ để lấy giá trị từ các cột
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    # Lấy dữ liệu cho các chỉ số thủ môn
    goalkeeper_data = [
        get_stat('gk_goals_against'),
        get_stat('gk_goals_against_per90'),
        get_stat('gk_shots_on_target_against'),
        get_stat('gk_saves'),
        get_stat('gk_save_pct'),
        get_stat('gk_wins'),
        get_stat('gk_ties'),
        get_stat('gk_losses'),
        get_stat('gk_clean_sheets'),
        get_stat('gk_clean_sheets_pct'),
        get_stat('gk_pens_att'),
        get_stat('gk_pens_allowed'),
        get_stat('gk_pens_saved'),
        get_stat('gk_pens_missed'),
        get_stat('gk_pens_save_pct')
    ]

    return goalkeeper_data
```

- process_shooting_data: Xử lý dữ liệu về sút bóng.

```
def process_shooting_data(player):
    # Hàm phụ để lấy giá trị từ cột
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    # Lấy dữ liệu cho các chỉ số về sút bóng
    shooting_data = [
        get_stat('goals'),
        get_stat('shots'),
        get_stat('shots_on_target'),
        get_stat('shots_on_target_pct'),
        get_stat('shots_per90'),
        get_stat('shots_on_target_per90'),
        get_stat('goals_per_shot'),
        get_stat('goals_per_shot_on_target'),
        get_stat('average_shot_distance'),
        get_stat('shots_free_kicks'),
        get_stat('pens_made'),
        get_stat('pens_att'),
        get_stat('xg'),
        get_stat('npxg'),
        get_stat('npxg_per_shot'),
        get_stat('xg_net'),
        get_stat('npxg_net')
    ]

    return shooting_data
```

- process_passing_data: Xử lý dữ liệu về chuyền bóng.

```python
# Hàm xử lý dữ liệu Passing
def process_passing_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    passing_data = [
        get_stat('passes_completed'),
        get_stat('passes'),
        get_stat('passes_pct'),
        get_stat('passes_total_distance'),
        get_stat('passes_progressive_distance'),
        get_stat('passes_completed_short'),
        get_stat('passes_short'),
        get_stat('passes_pct_short'),
        get_stat('passes_completed_medium'),
        get_stat('passes_medium'),
        get_stat('passes_pct_medium'),
        get_stat('passes_completed_long'),
        get_stat('passes_long'),
        get_stat('passes_pct_long'),
        get_stat('assists'),
        get_stat('xg_assist'),
        get_stat('pass_xa'),
        get_stat('xg_assist_net'),
        get_stat('assisted_shots'),
        get_stat('passes_into_final_third'),
        get_stat('passes_into_penalty_area'),
        get_stat('crosses_into_penalty_area'),
        get_stat('progressive_passes')
    ]
    return passing_data
```

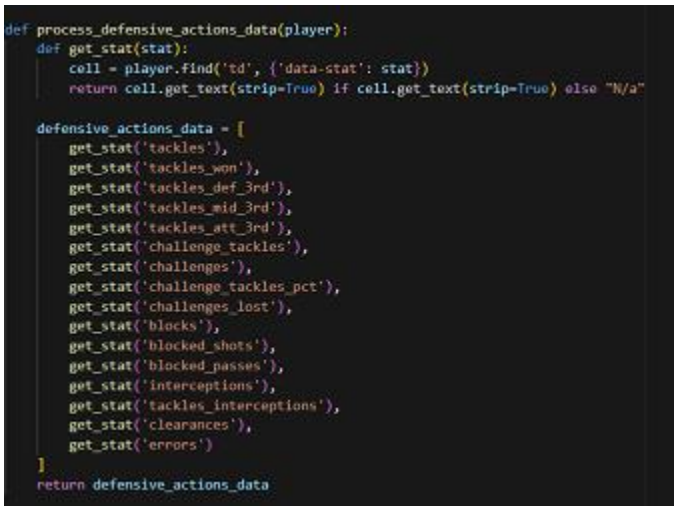
- process_pass_types_data: Xử lý dữ liệu về loại đường chuyền.

```python
def process_pass_types_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    pass_types_data = [
        get_stat('passes_live'),
        get_stat('passes_dead'),
        get_stat('passes_free_kicks'),
        get_stat('through_balls'),
        get_stat('passes_switches'),
        get_stat('crosses'),
        get_stat('throw_ins'),
        get_stat('corner_kicks'),
        get_stat('corner_kicks_in'),
        get_stat('corner_kicks_out'),
        get_stat('corner_kicks_straight'),
        get_stat('passes_completed'),
        get_stat('passes_offsides'),
        get_stat('passes_blocked')
    ]
    return pass_types_data
```

- process_goal_and_shot_creation_data: Xử lý dữ liệu về tạo cơ hội ghi bàn và sút bóng.

```python
def process_goal_and_shot_creation_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    goal_and_shot_creation_data = [
        get_stat('sca'),
        get_stat('sca_per90'),
        get_stat('sca_passes_live'),
        get_stat('sca_passes_dead'),
        get_stat('sca_take_ons'),
        get_stat('sca_shots'),
        get_stat('sca_fouled'),
        get_stat('sca_defense'),
        get_stat('gca'),
        get_stat('gca_per90'),
        get_stat('gca_passes_live'),
        get_stat('gca_passes_dead'),
        get_stat('gca_take_ons'),
        get_stat('gca_shots'),
        get_stat('gca_fouled'),
        get_stat('gca_defense')
    ]
    return goal_and_shot_creation_data
```

- process_defensive_actions_data: Xử lý dữ liệu về hành động phòng thủ.



```
def process_defensive_actions_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    defensive_actions_data = [
        get_stat('tackles'),
        get_stat('tackles_won'),
        get_stat('tackles_def_3rd'),
        get_stat('tackles_mid_3rd'),
        get_stat('tackles_att_3rd'),
        get_stat('challenge_tackles'),
        get_stat('challenges'),
        get_stat('challenge_tackles_pct'),
        get_stat('challenges_lost'),
        get_stat('blocks'),
        get_stat('blocked_shots'),
        get_stat('blocked_passes'),
        get_stat('interceptions'),
        get_stat('tackles_interceptions'),
        get_stat('clearances'),
        get_stat('errors')
    ]
    return defensive_actions_data
```

- process_possession_data: Xử lý dữ liệu về kiểm soát bóng.

```
# Hàm xử lý dữ liệu Possession
def process_possession_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    possession_data = [
        get_stat('touches'),
        get_stat('touches_def_pen_area'),
        get_stat('touches_def_3rd'),
        get_stat('touches_mid_3rd'),
        get_stat('touches_att_3rd'),
        get_stat('touches_att_pen_area'),
        get_stat('touches_live_ball'),
        get_stat('take_ons'),
        get_stat('take_ons_won'),
        get_stat('take_ons_won_pct'),
        get_stat('take_ons_tackled'),
        get_stat('take_ons_tackled_pct'),
        get_stat('carries'),
        get_stat('carries_distance'),
        get_stat('carries_progressive_distance'),
        get_stat('progressive_carries'),
        get_stat('carries_into_final_third'),
        get_stat('carries_into_penalty_area'),
        get_stat('miscontrols'),
        get_stat('dispossessed'),
        get_stat('passes_received'),
        get_stat('progressive_passes_received')
    ]
    return possession_data
```

- process_playing_time_data: Xử lý dữ liệu về thời gian thi đấu.

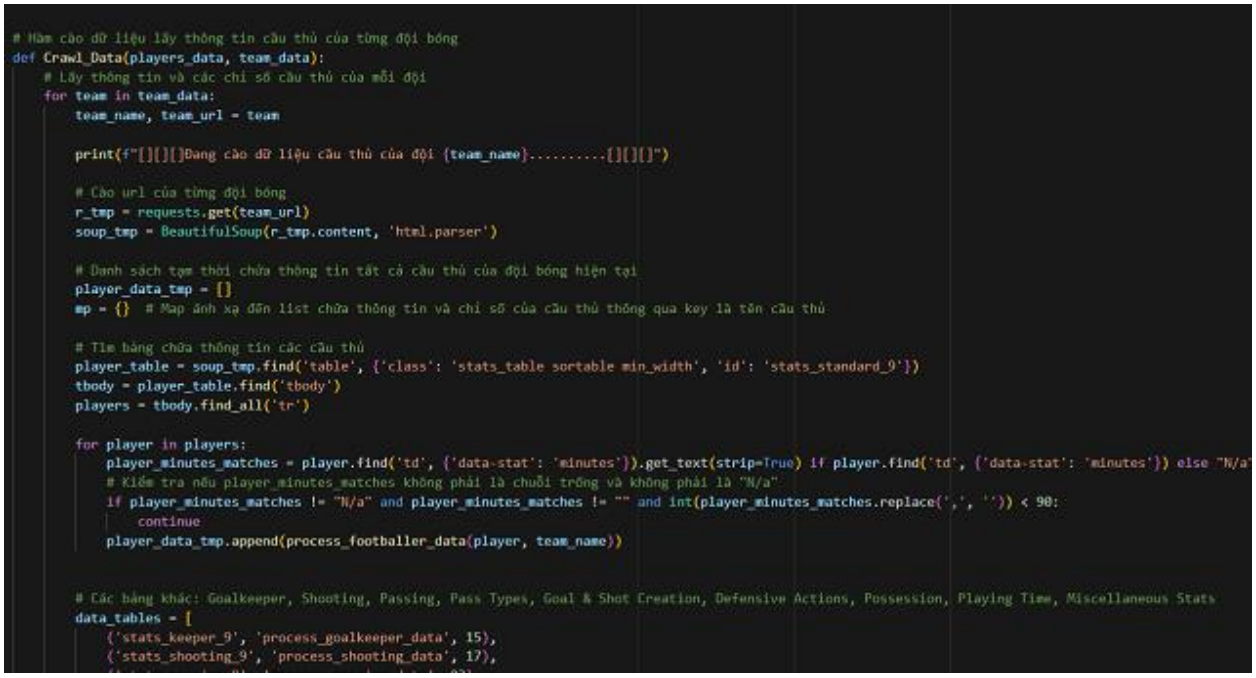
```
# Hàm xử lý dữ liệu Playing Time
def process_playing_time_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    playing_time_data = [
        get_stat('games_starts'),
        get_stat('minutes_per_start'),
        get_stat('games_complete'),
        get_stat('games_subs'),
        get_stat('minutes_per_sub'),
        get_stat('unused_subs'),
        get_stat('points_per_game'),
        get_stat('on_goals_for'),
        get_stat('on_goals_against'),
        get_stat('on_xg_for'),
        get_stat('on_xg_against')
    ]
    return playing_time_data
```

- process_miscellaneous_stats_data: Xử lý các thống kê khác.

```python
# Hàm xử lý dữ liệu Miscellaneous Stats
def process_miscellaneous_stats_data(player):
    def get_stat(stat):
        cell = player.find('td', {'data-stat': stat})
        return cell.get_text(strip=True) if cell.get_text(strip=True) else "N/a"

    miscellaneous_stats_data = [
        get_stat('fouls'),
        get_stat('fouled'),
        get_stat('offsides'),
        get_stat('crosses'),
        get_stat('own_goals'),
        get_stat('ball_recoveries'),
        get_stat('aerials_won'),
        get_stat('aerials_lost'),
        get_stat('aerials_won_pct')
    ]
    return miscellaneous_stats_data
```

- Mỗi hàm trên đều có cấu trúc tương tự, sử dụng hàm get_stat để trích xuất dữ liệu từ các cột tương ứng và trả về danh sách các chỉ số.

3.3. Hàm Crawl_Data

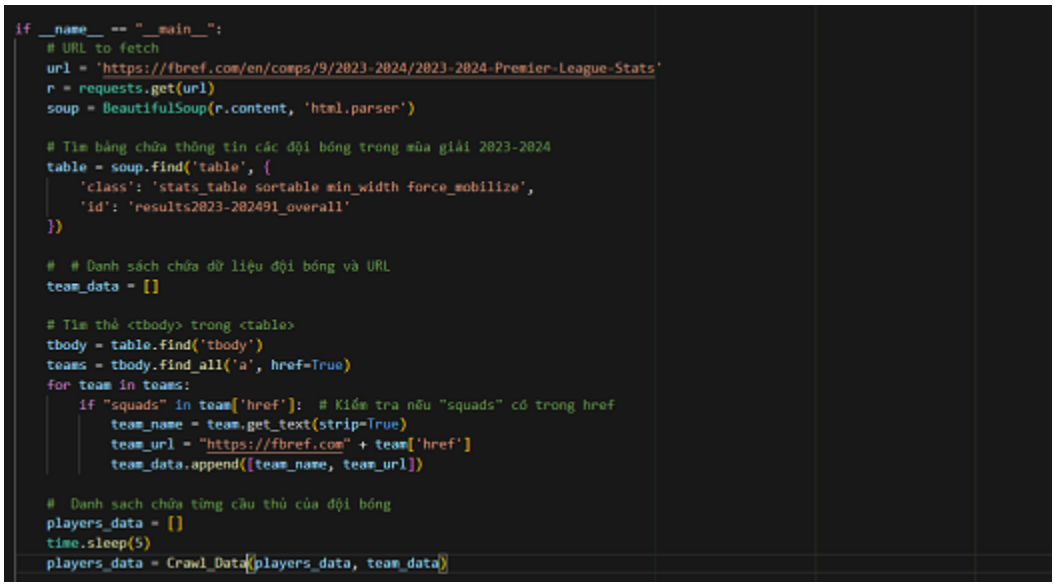


```python
# Hàm cào dữ liệu lấy thông tin cầu thủ của từng đội bóng
def Crawl_Data(players_data, team_data):
    # Lấy thông tin và các chỉ số cầu thủ của mỗi đội
    for team in team_data:
        team_name, team_url = team

        print(f"[][][]Đang cào dữ liệu cầu thủ của đội {team_name}.....[][][]")

        # Cào url của từng đội bóng
        r_tmp = requests.get(team_url)
        soup_tmp = BeautifulSoup(r_tmp.content, 'html.parser')

        # Danh sách tạm thời chứa thông tin tất cả cầu thủ của đội bóng hiện tại
        player_data_tmp = []
        mp = {}  # Map ánh xạ đến list chứa thông tin và chỉ số của cầu thủ thông qua key là tên cầu thủ

        # Tìm bảng chứa thông tin các cầu thủ
        player_table = soup_tmp.find('table', {'class': 'stats_table sortable min_width', 'id': 'stats_standard_9'})
        tbody = player_table.find('tbody')
        players = tbody.find_all('tr')

        for player in players:
            player_minutes_matches = player.find('td', {'data-stat': 'minutes'}).get_text(strip=True) if player.find('td', {'data-stat': 'minutes'}) else "N/a"
            # Kiểm tra nếu player_minutes_matches không phải là chuỗi trống và không phải là "N/a"
            if player_minutes_matches != "N/a" and player_minutes_matches != "" and int(player_minutes_matches.replace(',', '')) < 90:
                continue
            player_data_tmp.append(process_footballer_data(player, team_name))


        # Các bảng khác: Goalkeeper, Shooting, Passing, Pass Types, Goal & Shot Creation, Defensive Actions, Possession, Playing Time, Miscellaneous Stats
        data_tables = [
            ('stats_keeper_9', 'process_goalkeeper_data', 15),
            ('stats_shooting_9', 'process_shooting_data', 17),
```

- Mục đích: Cào dữ liệu cho từng đội bóng và tổng hợp thông tin của tất cả cầu thủ.
- Tham số:
- players_data: Danh sách lưu trữ dữ liệu của tất cả cầu thủ.
- team_data: Danh sách chứa tên và URL của các đội bóng.
- Chức năng:
- Lặp qua từng đội bóng trong team_data.
- Gửi yêu cầu HTTP đến trang thống kê của đội bóng.

- Tìm và phân tích các bảng dữ liệu chứa thông tin cầu thủ.
- Sử dụng các hàm xử lý dữ liệu tương ứng để trích xuất thông tin.
- Tổng hợp dữ liệu của cầu thủ vào players_data.

3.4. Phần chương trình chính (if __name__ == "__main__":)



```
if __name__ == "__main__":
    # URL to fetch
    url = 'https://fbref.com/en/comps/9/2023-2024/2023-2024-Premier-League-Stats'
    r = requests.get(url)
    soup = BeautifulSoup(r.content, 'html.parser')

    # Tìm bảng chứa thông tin các đội bóng trong mùa giải 2023-2024
    table = soup.find('table', {
        'class': 'stats_table sortable min_width force_mobilize',
        'id': 'results2023-202491_overall'
    })

    #  # Danh sách chứa dữ liệu đội bóng và URL
    team_data = []

    # Tìm thẻ <tbody> trong <table>
    tbody = table.find('tbody')
    teams = tbody.find_all('a', href=True)
    for team in teams:
        if "squads" in team['href']:  # Kiểm tra nếu "squads" có trong href
            team_name = team.get_text(strip=True)
            team_url = "https://fbref.com" + team['href']
            team_data.append([team_name, team_url])

    #  Danh sach chứa từng cầu thủ của đội bóng
    players_data = []
    time.sleep(5)
    players_data = Crawl_Data(players_data, team_data)
```

- Gửi yêu cầu đến trang thống kê giải Ngoại hạng Anh mùa 2023-2024.
- Trích xuất tên và URL của các đội bóng.
- Khởi tạo danh sách players_data để lưu trữ dữ liệu cầu thủ.
- Gọi hàm Crawl_Data_For_Each_Team để cào dữ liệu.
- Sắp xếp dữ liệu theo tên cầu thủ và tuổi giảm dần.
- Chuyển dữ liệu thành DataFrame và lưu vào tệp CSV results.csv.

CÂU 2: Tìm top 3 cầu thủ có điểm cao nhất và thấp nhất ở mỗi chỉ số. Tìm trung vị của mỗi chỉ số. Tìm trung bình và độ lệch chuẩn của mỗi chỉ số cho các cầu thủ trong toàn giải và của mỗi đội. Vẽ historgram phân bố của mỗi chỉ số của các cầu thủ trong toàn giải và mỗi đội. Tìm đội bóng có chỉ số điểm số cao nhất ở mỗi chỉ số. Theo bạn đội nào có phong độ tốt nhất giải ngoại Hạng Anh mùa 2023-2024.

1. Giới thiệu

- Mã nguồn được cung cấp thực hiện việc phân tích dữ liệu cầu thủ bóng đá từ tệp CSV đã được thu thập trước đó (trong trường hợp này là results.csv). Mục đích của chương trình là:
- Xác định Top 3 cầu thủ có giá trị cao nhất và thấp nhất cho mỗi chỉ số hiệu suất.
- Tính toán các thống kê (trung vị, trung bình, độ lệch chuẩn) cho toàn bộ giải đấu và từng đội bóng.

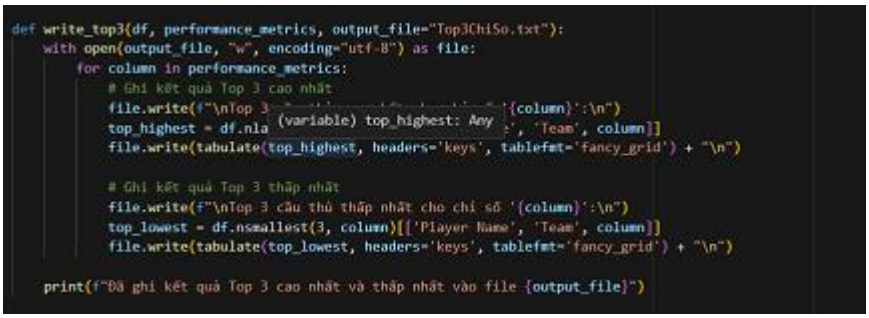
- Vẽ biểu đồ histogram cho từng chỉ số hiệu suất, cả cho toàn giải và từng đội.
- Xác định đội bóng có phong độ tốt nhất dựa trên các chỉ số hiệu suất.

2. Thư viện và công cụ sử dụng

- sys: Thư viện hỗ trợ thao tác với hệ thống, ở đây dùng để thiết lập mã hóa UTF-8 cho đầu ra.
- pandas: Thư viện mạnh mẽ để xử lý và phân tích dữ liệu dạng bảng.
- numpy: Thư viện hỗ trợ tính toán số học hiệu quả với mảng và ma trận lớn.
- tabulate: Thư viện giúp hiển thị dữ liệu dưới dạng bảng đẹp mắt trong console hoặc ghi vào file.
- matplotlib.pyplot và seaborn: Thư viện vẽ biểu đồ và trực quan hóa dữ liệu.
- collections.Counter: Dùng để đếm tần suất xuất hiện của các phần tử.
- os và time: Thư viện hỗ trợ thao tác với hệ thống tệp và thời gian.

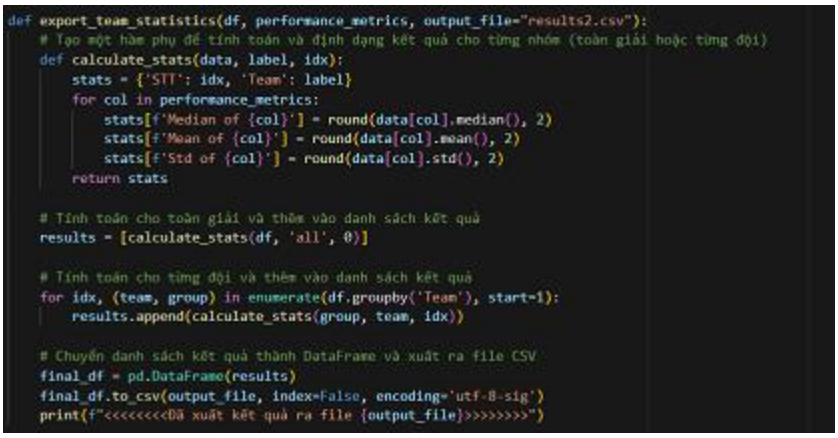
3. Cấu trúc và chức năng của mã nguồn

3.1. Hàm write_top3



- Mục đích: Xác định và ghi vào file Top 3 cầu thủ có giá trị cao nhất và thấp nhất cho mỗi chỉ số hiệu suất.
- Chi tiết:
  - Tham số:
    - df: DataFrame chứa dữ liệu cầu thủ.
    - performance_metrics: Danh sách các cột chỉ số hiệu suất cần phân tích.
    - output_file: Tên file đầu ra để ghi kết quả (mặc định là "Top3ChiSo.txt").
  - Hoạt động:
    - Duyệt qua từng chỉ số trong performance_metrics.
    - Sử dụng nlargest(3, column) và nsmallest(3, column) để tìm Top 3 cầu thủ cao nhất và thấp nhất cho từng chỉ số.
    - Ghi kết quả vào file dưới dạng bảng sử dụng tabulate.

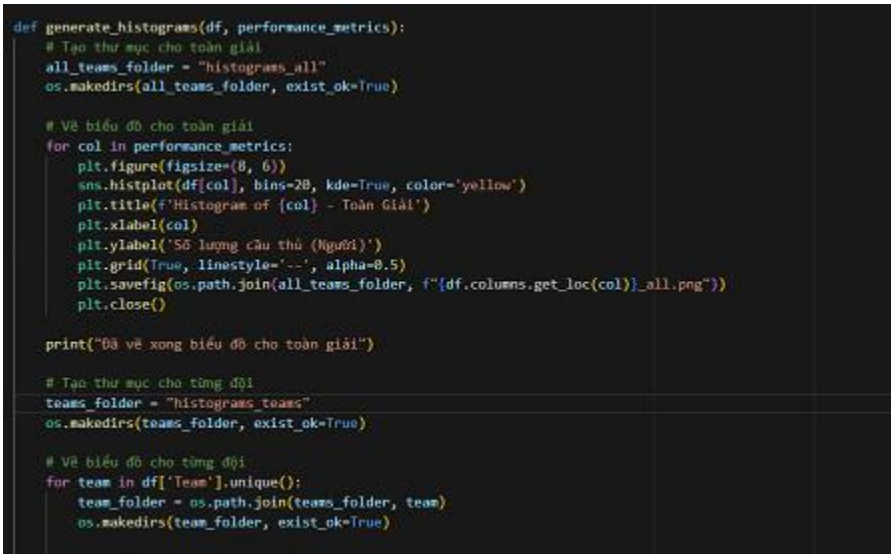
3.2. Hàm export_team_statistics



```python
def export_team_statistics(df, performance_metrics, output_file="results2.csv"):
    # Tạo một hàm phụ để tính toán và định dạng kết quả cho từng nhóm (toàn giải hoặc từng đội)
    def calculate_stats(data, label, idx):
        stats = {'STT': idx, 'Team': label}
        for col in performance_metrics:
            stats[f'Median of {col}'] = round(data[col].median(), 2)
            stats[f'Mean of {col}'] = round(data[col].mean(), 2)
            stats[f'Std of {col}'] = round(data[col].std(), 2)
        return stats

    # Tính toán cho toàn giải và thêm vào danh sách kết quả
    results = [calculate_stats(df, 'all', 0)]

    # Tính toán cho từng đội và thêm vào danh sách kết quả
    for idx, (team, group) in enumerate(df.groupby('Team'), start=1):
        results.append(calculate_stats(group, team, idx))

    # Chuyển danh sách kết quả thành DataFrame và xuất ra file CSV
    final_df = pd.DataFrame(results)
    final_df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"<<<<<<<<<Đã xuất kết quả ra file {output_file}>>>>>>>>>")
```

- Mục đích: Tính toán các thống kê (trung vị, trung bình, độ lệch chuẩn) cho toàn giải và từng đội bóng, sau đó xuất ra file CSV.
- Chi tiết:
    - Tham số:
        - df: DataFrame chứa dữ liệu cầu thủ.
        - performance_metrics: Danh sách các cột chỉ số hiệu suất cần phân tích.
        - output_file: Tên file CSV đầu ra (mặc định là "results2.csv").
    - Hoạt động:
        - Định nghĩa hàm phụ calculate_stats để tính toán thống kê cho từng nhóm dữ liệu.
        - Tính toán thống kê cho toàn giải và lưu vào danh sách results.
        - Nhóm dữ liệu theo Team và tính toán thống kê cho từng đội, sau đó thêm vào results.
        - Chuyển results thành DataFrame và xuất ra file CSV.

3.3. Hàm generate_histograms



```python
def generate_histograms(df, performance_metrics):
    # Tạo thư mục cho toàn giải
    all_teams_folder = "histograms_all"
    os.makedirs(all_teams_folder, exist_ok=True)

    # Vẽ biểu đồ cho toàn giải
    for col in performance_metrics:
        plt.figure(figsize=(8, 6))
        sns.histplot(df[col], bins=20, kde=True, color='yellow')
        plt.title(f'Histogram of {col} - Toàn Giải')
        plt.xlabel(col)
        plt.ylabel('Số lượng cầu thủ (Người)')
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.savefig(os.path.join(all_teams_folder, f"{df.columns.get_loc(col)}_all.png"))
        plt.close()

    print("Đã vẽ xong biểu đồ cho toàn giải")

    # Tạo thư mục cho từng đội
    teams_folder = "histograms_teams"
    os.makedirs(teams_folder, exist_ok=True)

    # Vẽ biểu đồ cho từng đội
    for team in df['Team'].unique():
        team_folder = os.path.join(teams_folder, team)
        os.makedirs(team_folder, exist_ok=True)
```

- Mục đích: Vẽ biểu đồ histogram cho từng chỉ số hiệu suất, cả cho toàn giải và từng đội bóng.
- Chi tiết:
    - Tham số:
        - df: DataFrame chứa dữ liệu cầu thủ.
        - performance_metrics: Danh sách các cột chỉ số hiệu suất cần phân tích.
    - Hoạt động:
        - Tạo thư mục histograms_all để lưu biểu đồ của toàn giải.
        - Duyệt qua từng chỉ số trong performance_metrics, sử dụng seaborn.histplot để vẽ biểu đồ và lưu vào thư mục.
        - Tạo thư mục histograms_teams để lưu biểu đồ của từng đội.
        - Duyệt qua từng đội bóng, tạo thư mục riêng cho mỗi đội.
        - Với mỗi đội, vẽ và lưu biểu đồ histogram cho từng chỉ số.

3.4. Hàm identify_best_teams

```
def identify_best_teams(df, performance_metrics):
    # Chuyển đổi các cột chỉ số thành dạng số nếu cần thiết
    df[performance_metrics] = df[performance_metrics].apply(pd.to_numeric, errors='coerce')

    # Tính giá trị trung bình của mỗi chỉ số cho từng đội
    team_averages = df.groupby('Team')[performance_metrics].mean()

    # Tìm đội có giá trị cao nhất cho mỗi chỉ số
    best_teams = [
        [stat, team_averages[stat].idxmax(), team_averages[stat].max()]
        for stat in performance_metrics
    ]

    # Hiển thị bảng kết quả chỉ số cao nhất cho mỗi chỉ số
    headers = ["Chỉ số", "Team", "Giá trị"]
    print(tabulate(best_teams, headers=headers, tablefmt="grid"))

    # Tính toán tần suất xuất hiện của từng đội trong các chỉ số cao nhất
    team_frequency = Counter(team for _, team, _ in best_teams)
    sorted_frequency = sorted(team_frequency.items(), key=lambda x: x[1], reverse=True)

    # Hiển thị bảng tần suất của các đội
    print("\nTần suất của từng đội bóng:")
    print(tabulate(sorted_frequency, headers=["Team", "Số lần"], tablefmt="grid"))
```

- Mục đích: Xác định đội bóng có phong độ tốt nhất dựa trên các chỉ số hiệu suất.
- Chi tiết:
    - Tham số:
        - df: DataFrame chứa dữ liệu cầu thủ.
        - performance_metrics: Danh sách các cột chỉ số hiệu suất cần phân tích.
    - Hoạt động:
        - Chuyển đổi các cột chỉ số thành dạng số (nếu chưa).
        - Tính giá trị trung bình của mỗi chỉ số cho từng đội bóng.
        - Tìm đội bóng có giá trị cao nhất cho mỗi chỉ số.
        - Sử dụng Counter để đếm tần suất xuất hiện của từng đội trong danh sách các chỉ số cao nhất.

- Hiển thị bảng kết quả và xác định đội bóng có phong độ tốt nhất dựa trên tần suất cao nhất.

3.5. Phần chương trình chính (if __name__ == "__main__":)

```
if __name__ == "__main__":
    df = pd.read_csv("results.csv")
    performance_metrics = df.columns[4:]
    df[performance_metrics] = df[performance_metrics].apply(pd.to_numeric, errors='coerce')

    write_top3(df, performance_metrics)
    export_team_statistics(df, performance_metrics)
    generate_histograms(df, performance_metrics)
    identify_best_teams(df, df.columns[8:])
```

- Đọc dữ liệu từ tệp results.csv vào DataFrame df.
- Xác định danh sách các chỉ số hiệu suất cần phân tích (từ cột thứ 5 trở đi).
- Chuyển đổi các cột chỉ số thành dạng số, bỏ qua lỗi nếu có.
- Gọi lần lượt các hàm:
- write_top3 để ghi kết quả Top 3 vào file.
- export_team_statistics để xuất thống kê ra file CSV.
- generate_histograms để vẽ và lưu biểu đồ.
- identify_best_teams để xác định đội bóng có phong độ tốt nhất**.**

CÂU 3: Sử dụng thuật toán K-means để phân loại các cầu thủ thành các nhóm có chỉ số giống nhau. Theo bạn thì nên phân loại cầu thủ thành bao nhiêu nhóm? Vì sao? Bạn có Nhận xét gì về kết quả. Sử dụng thuật toán PCA, giảm số chiều dữ liệu xuống 2 chiều, vẽ hình phân cụm các điểm dữ liệu trên mặt 2D. Viết chương trình python vẽ biểu đồ rada (radar chart) so sánh cầu thủ.

1. **Sử dụng thuật toán K-means để phân loại các cầu thủ thành các nhóm có chỉ số giống nhau. Theo bạn thì nên phân loại cầu thủ thành bao nhiêu nhóm? Vì sao? Bạn có Nhận xét gì về kết quả. Sử dụng thuật toán PCA, giảm số chiều dữ liệu xuống 2 chiều, vẽ hình phân cụm các điểm dữ liệu trên mặt 2D.**
   1. **Giới thiệu**
      - Mã nguồn được cung cấp thực hiện việc phân cụm dữ liệu cầu thủ bóng đá bằng thuật toán K-means clustering. Dữ liệu được lấy từ tệp results.csv, bao gồm các thông tin và chỉ số hiệu suất của cầu thủ. Mục đích chính của chương trình là:
      - Đọc và chuẩn bị dữ liệu cho quá trình phân cụm.
      - Chuẩn hóa và giảm chiều dữ liệu để thuận tiện cho việc phân cụm và trực quan hóa.
      - Thực hiện thuật toán K-means để phân chia cầu thủ thành các cụm dựa trên đặc trưng của họ.

      Trực quan hóa kết quả phân cụm bằng biểu đồ scatter plot.

      2. Thư viện và công cụ sử dụng

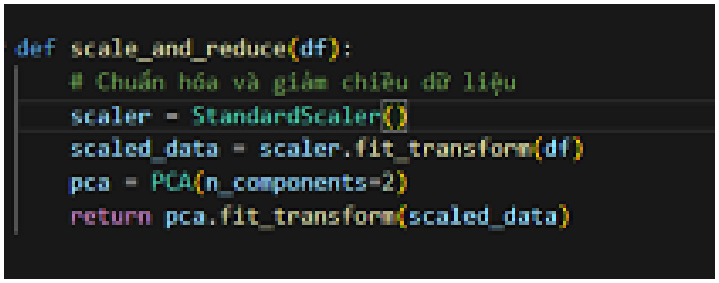
- pandas: Thư viện hỗ trợ thao tác và quản lý dữ liệu dưới dạng DataFrame.
- numpy: Thư viện hỗ trợ tính toán số học hiệu quả với mảng và ma trận lớn.
- sklearn.preprocessing.StandardScaler: Dùng để chuẩn hóa dữ liệu.
- sklearn.cluster.KMeans: Thuật toán K-means clustering (mặc dù trong mã nguồn, thuật toán K-means được tự triển khai, không sử dụng trực tiếp từ sklearn).
- sklearn.decomposition.PCA: Giảm chiều dữ liệu bằng Phân tích Thành phần Chính (Principal Component Analysis).
- matplotlib.pyplot: Thư viện vẽ biểu đồ và trực quan hóa dữ liệu.

3. Cấu trúc và chức năng của mã nguồn

3.1. Hàm initialize_data()

- Mục đích: Đọc dữ liệu từ tệp CSV và chuẩn bị dữ liệu cho quá trình xử lý tiếp theo.
- Chi tiết:
    - Đọc dữ liệu: Sử dụng pd.read_csv('results.csv') để đọc dữ liệu từ tệp CSV vào DataFrame df.
    - Lọc dữ liệu số: Sử dụng df.select_dtypes(exclude=['object']) để chỉ giữ lại các cột có kiểu dữ liệu số, loại bỏ các cột kiểu chuỗi (như tên cầu thủ, quốc tịch, v.v.).
    - Xử lý giá trị thiếu: Sử dụng df.fillna(df.mean()) để thay thế các giá trị thiếu (NaN) bằng giá trị trung bình của cột tương ứng.
    - Trả về: DataFrame df đã được chuẩn bị.
- Nhận xét:
    - Việc loại bỏ các cột không phải số giúp tránh lỗi khi thực hiện các phép tính toán số học.
    - Thay thế giá trị thiếu bằng trung bình giúp dữ liệu đầy đủ hơn cho quá trình phân cụm.

3.2. Hàm scale_and_reduce(df)



- Mục đích: Chuẩn hóa dữ liệu và giảm chiều dữ liệu để thuận tiện cho phân cụm và trực quan hóa.
- Chi tiết:
  - Chuẩn hóa dữ liệu:
    - Sử dụng StandardScaler() để chuẩn hóa dữ liệu, biến đổi dữ liệu sao cho có trung bình bằng 0 và độ lệch chuẩn bằng 1.
    - Áp dụng scaler.fit_transform(df) để chuẩn hóa DataFrame df, kết quả là mảng scaled_data.
  - Giảm chiều dữ liệu:
    - Sử dụng PCA(n_components=2) để giảm chiều dữ liệu xuống còn 2 thành phần chính.
    - Áp dụng pca.fit_transform(scaled_data) để thực hiện PCA trên dữ liệu đã chuẩn hóa, kết quả là mảng dữ liệu data có 2 chiều.
  - Trả về: Mảng dữ liệu data sau khi đã chuẩn hóa và giảm chiều.
- Nhận xét:
  - Việc chuẩn hóa dữ liệu là cần thiết để các đặc trưng có đơn vị đo khác nhau không ảnh hưởng đến kết quả phân cụm.
  - Giảm chiều dữ liệu xuống 2 chiều giúp việc trực quan hóa kết quả phân cụm dễ dàng hơn trên biểu đồ 2D.

3.3. Hàm perform_kmeans(data, num_clusters=5, max_iter=100)

```
def perform_kmeans(data, num_clusters=5, max_iter=100):
    # Chuyển đổi dữ liệu thành mảng NumPy nếu là DataFrame
    if isinstance(data, pd.DataFrame):
        data = data.values

    # Khởi tạo ngẫu nhiên các tâm cụm
    centroids = data[np.random.choice(data.shape[0], num_clusters, replace=False)]

    for step in range(max_iter):
        # Tính khoảng cách từ mỗi điểm đến tất cả tâm cụm và gán cụm
        distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
        clusters = np.argmin(distances, axis=1)

        # Tính tâm cụm mới bằng cách lấy trung bình cộng của các điểm trong mỗi cụm
        new_centroids = np.array([data[clusters == i].mean(axis=0) for i in range(num_clusters)])

        # Kiểm tra sự hội tụ: nếu tâm cụm không thay đổi thì dừng lặp
        if np.allclose(centroids, new_centroids):
            break

        centroids = new_centroids

    return centroids, clusters
```

- Mục đích: Thực hiện thuật toán K-means clustering trên dữ liệu để phân chia cầu thủ thành các cụm.
- Chi tiết:
  - Kiểm tra và chuyển đổi dữ liệu:
    - Nếu data là một DataFrame, chuyển đổi thành mảng NumPy bằng data.values.
  - Khởi tạo tâm cụm ban đầu:
    - Chọn ngẫu nhiên num_clusters (mặc định là 5) điểm dữ liệu từ data để làm tâm cụm ban đầu.

        - Sử dụng np.random.choice để chọn chỉ số các điểm dữ liệu.
    - Vòng lặp thuật toán K-means:
        - Bước gán cụm:
            - Tính khoảng cách Euclidean từ mỗi điểm đến tất cả các tâm cụm hiện tại.
            - Gán mỗi điểm dữ liệu vào cụm có khoảng cách gần nhất (sử dụng np.argmin).
        - Bước cập nhật tâm cụm:
            - Tính toán tâm cụm mới bằng cách lấy trung bình cộng của các điểm trong mỗi cụm.
        - Kiểm tra hội tụ:
            - Sử dụng np.allclose để kiểm tra xem tâm cụm mới có khác biệt đáng kể so với tâm cụm cũ hay không.
            - Nếu tâm cụm không thay đổi (hoặc thay đổi rất nhỏ), thuật toán hội tụ và dừng lặp.
        - Cập nhật tâm cụm:
            - Nếu chưa hội tụ, cập nhật tâm cụm với giá trị mới và tiếp tục vòng lặp.
    - Trả về:
        - centroids: Mảng chứa tọa độ các tâm cụm cuối cùng.
        - clusters: Mảng chứa chỉ số cụm mà mỗi điểm dữ liệu thuộc về.
- Nhận xét:
    - Thuật toán K-means được tự triển khai mà không sử dụng trực tiếp từ thư viện sklearn.cluster.KMeans.
    - Số lượng cụm num_clusters có thể được điều chỉnh tùy theo nhu cầu phân tích.

3.4. Hàm visualize_clusters(data, centroids, clusters)

```
def visualize_clusters(data, centroids, clusters):
    # Vẽ biểu đồ các cụm với phong cách khác
    plt.figure(figsize=(10, 8))
    unique_clusters = np.unique(clusters)
    colors = plt.cm.get_cmap('tab10', len(unique_clusters))  # Sử dụng bảng màu 'tab10' cho đa dạng màu sắc

    # Vẽ điểm dữ liệu với màu sắc và nhãn tương ứng với từng cụm
    for i, color in zip(unique_clusters, colors.colors):
        cluster_data = data[clusters == i]
        plt.scatter(cluster_data[:, 0], cluster_data[:, 1], color=color, label=f'Cluster {i}', alpha=0.6, edgecolor='w')

    # Vẽ các tâm cụm với ký hiệu và màu đậm hơn
    for idx, centroid in enumerate(centroids):
        plt.scatter(*centroid, color='black', marker='X', s=250, edgecolor='w', linewidth=2)
        plt.text(centroid[0], centroid[1], f'Center {idx}', fontsize=12, weight='bold', ha='center')

    plt.title('Enhanced Visualization of K-means Clustering')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(title='Cluster ID')
    plt.grid(True)
    plt.show()
```

- Mục đích: Trực quan hóa kết quả phân cụm bằng biểu đồ scatter plot.
- Chi tiết:
    - Khởi tạo biểu đồ:

        - Tạo một figure với kích thước 10x8 inch.
        - Lấy danh sách các cụm duy nhất từ clusters.
        - Sử dụng plt.cm.get_cmap('tab10', len(unique_clusters)) để tạo một bảng màu với nhiều màu sắc khác nhau cho các cụm.
    - Vẽ các điểm dữ liệu:
        - Lặp qua từng cụm và màu sắc tương ứng.
        - Lấy các điểm dữ liệu thuộc cụm hiện tại.
        - Sử dụng plt.scatter để vẽ các điểm dữ liệu, với màu sắc và nhãn tương ứng.
        - Sử dụng các tham số alpha=0.6 (độ trong suốt) và edgecolor='w' (màu viền trắng) để cải thiện giao diện.
    - Vẽ tâm cụm:
        - Lặp qua các tâm cụm và vẽ chúng trên biểu đồ với ký hiệu 'X', kích thước lớn hơn và màu đen.
        - Thêm nhãn cho mỗi tâm cụm bằng plt.text.
    - Cài đặt biểu đồ:
        - Đặt tiêu đề, nhãn trục x và y.
        - Thêm chú giải (legend) với tiêu đề 'Cluster ID'.
        - Thêm lưới (grid) để dễ quan sát.
    - Hiển thị biểu đồ:
        - Sử dụng plt.show() để hiển thị biểu đồ.
- Nhận xét:
    - Biểu đồ trực quan giúp quan sát sự phân bố của các cụm và mối quan hệ giữa các điểm dữ liệu.
    - Việc giảm chiều dữ liệu xuống 2D bằng PCA trước đó là cần thiết để vẽ biểu đồ 2D.

3.5. Phần chương trình chính (if __name__ == "__main__":)

```
if __name__ == "__main__":
    df = initialize_data()
    data = scale_and_reduce(df)
    centroids, clusters = perform_kmeans(data)
    visualize_clusters(data, centroids, clusters)
```

- Khởi tạo dữ liệu:
    - Gọi hàm initialize_data() để đọc và chuẩn bị dữ liệu, kết quả là DataFrame df.
    - Chuẩn hóa và giảm chiều dữ liệu:
    - Gọi hàm scale_and_reduce(df) để chuẩn hóa và giảm chiều dữ liệu, kết quả là mảng data.
- Thực hiện phân cụm K-means:
    - Gọi hàm perform_kmeans(data) để thực hiện phân cụm, kết quả là centroids và clusters.

- Trực quan hóa kết quả phân cụm:
- Gọi hàm visualize_clusters(data, centroids, clusters) để vẽ biểu đồ kết quả.

2. **Viết chương trình python vẽ biểu đồ rada (radar chart) so sánh cầu thủ. Với đầu vào như sau:**

+ python radarChartPlot.py --p1 <player Name 1> --p2 <player Name 2> --Attribute <att1,att2,...,att_n>
+ --p1: là tên cầu thủ thứ nhất
+ --p2: là tên cầu thủ thứ hai
+ --Attribute: là danh sách các chỉ số cần so sánh

1. Giới thiệu

- Mã nguồn được cung cấp là một script Python nhằm mục đích so sánh hai cầu thủ bóng đá dựa trên các chỉ số hiệu suất cụ thể bằng cách sử dụng biểu đồ radar. Biểu đồ radar là một công cụ trực quan hóa dữ liệu hiệu quả, cho phép hiển thị và so sánh nhiều biến số trên cùng một đồ thị. Điều này đặc biệt hữu ích trong lĩnh vực thể thao, nơi việc đánh giá hiệu suất của cầu thủ dựa trên nhiều chỉ số khác nhau.

2. Thư viện và công cụ sử dụng

- pandas: Thư viện mạnh mẽ để thao tác và phân tích dữ liệu dạng bảng.
- numpy: Thư viện hỗ trợ tính toán số học hiệu quả với mảng và ma trận lớn.
- matplotlib.pyplot: Thư viện vẽ biểu đồ và trực quan hóa dữ liệu.
- argparse: Thư viện để phân tích các tham số dòng lệnh khi chạy script Python.

3. Cấu trúc và chức năng của mã nguồn

3.1. Hàm enhanced_radar_chart

```python
def enhanced_radar_chart(data, player1, player2, attributes):
    # Số lượng các thuộc tính
    num_attributes = len(attributes)

    # Trích xuất dữ liệu cho hai cầu thủ
    values1 = data[data['Player Name'] == player1][attributes].values.flatten()
    values2 = data[data['Player Name'] == player2][attributes].values.flatten()

    # Góc cho mỗi trục trong biểu đồ radar
    angles = np.linspace(0, 2 * np.pi, num_attributes, endpoint=False).tolist()

    # Đóng vòng dữ liệu
    values1 = np.concatenate((values1, [values1[0]]))
    values2 = np.concatenate((values2, [values2[0]]))
    angles += angles[:1]

    # Khởi tạo biểu đồ radar
    fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(polar=True))
    ax.plot(angles, values1, 'o-', linewidth=2, label=player1, color='red', markersize=10, markerfacecolor='yellow')
    ax.fill(angles, values1, alpha=0.1, color='red')
    ax.plot(angles, values2, 'd-', linewidth=2, label=player2, color='blue', markersize=10, markerfacecolor='lightblue')
    ax.fill(angles, values2, alpha=0.1, color='blue')

    # Cài đặt nhãn cho mỗi trục, cải tiến với font chữ lớn hơn
```

- Mục đích: Tạo biểu đồ radar so sánh hai cầu thủ dựa trên các thuộc tính được chỉ định.
- Chi tiết:
  - Tham số:
    - data: DataFrame chứa dữ liệu cầu thủ.
    - player1: Tên cầu thủ thứ nhất.
    - player2: Tên cầu thủ thứ hai.
    - attributes: Danh sách các thuộc tính (chỉ số) để so sánh.
  - Các bước thực hiện:
    - Xác định số lượng thuộc tính:
      - attributes = len(attributes).
    - Trích xuất dữ liệu cho hai cầu thủ:
      - Sử dụng pandas để lọc dữ liệu của từng cầu thủ và lấy các giá trị của các thuộc tính cần so sánh.
      - values1 và values2 chứa mảng các giá trị thuộc tính của player1 và player2.
      - Tính toán góc cho mỗi trục trong biểu đồ radar:
        - Sử dụng np.linspace để tạo mảng các góc chia đều từ 0 đến 2π.
        - angles là danh sách các góc tương ứng với mỗi thuộc tính.
      - Đóng vòng dữ liệu:
        - Để biểu đồ radar được khép kín, cần thêm giá trị đầu tiên vào cuối mảng giá trị và góc.
        - values1 và values2 được nối thêm giá trị đầu tiên của chúng.
        - angles cũng được nối thêm góc đầu tiên.
      - Khởi tạo biểu đồ radar:
        - Tạo một subplot với tham số polar=True để vẽ biểu đồ radar.
        - Vẽ đường và tô màu cho khu vực dưới đường cho mỗi cầu thủ:
        - ax.plot vẽ đường nối các điểm dữ liệu.
        - ax.fill tô màu khu vực bên trong đường.
      - Cài đặt nhãn cho mỗi trục và cải thiện giao diện:
        - Sử dụng ax.set_xticks và ax.set_xticklabels để đặt nhãn cho các trục tương ứng với các thuộc tính.
        - Tăng kích thước font và đặt độ đậm để nhãn dễ đọc hơn.
      - Thêm đường lưới và các chi tiết khác:
        - Bật lưới cho trục x và y để biểu đồ rõ ràng hơn.
        - Thêm tiêu đề và chú thích (legend) cho biểu đồ.

- Hiển thị biểu đồ:
    - Sử dụng plt.show() để hiển thị biểu đồ radar.

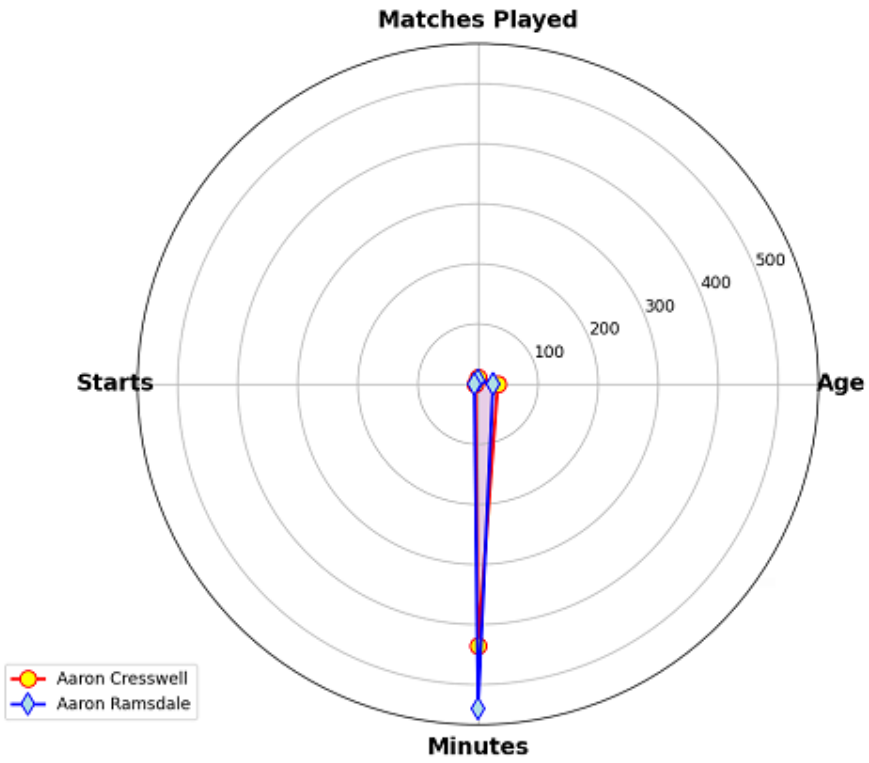
3.2. Phần chương trình chính (if __name__ == '__main__':)

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='So sánh hai cầu thủ bằng biểu đồ radar.')
    parser.add_argument('--p1', type=str, required=True, help='Tên cầu thủ thứ nhất')
    parser.add_argument('--p2', type=str, required=True, help='Tên cầu thủ thứ hai')
    parser.add_argument('--Attribute', type=str, required=True, help='Danh sách các thuộc tính cách nhau bằng dấu phẩy')

    args = parser.parse_args()

    data = pd.read_csv('results.csv')  # Đổi đường dẫn file dữ liệu
    attributes = args.Attribute.split(',')

    enhanced_radar_chart(data, args.p1, args.p2, attributes)
```

- Mục đích: Xử lý các tham số đầu vào từ dòng lệnh và gọi hàm enhanced_radar_chart để vẽ biểu đồ.
- Chi tiết:
    - Sử dụng argparse để phân tích tham số dòng lệnh:
        - tượng ArgumentParser với mô tả về chương trình.
    - Thêm các tham số cần thiết:
        - --p1: Tên cầu thủ thứ nhất.
        - --p2: Tên cầu thủ thứ hai.
        - --Attribute: Danh sách các thuộc tính, cách nhau bằng dấu phẩy.
    - Phân tích các tham số và đọc dữ liệu:
        - Gọi args = parser.parse_args() để lấy các tham số từ dòng lệnh.
        - Đọc dữ liệu từ tệp results.csv bằng pd.read_csv('results.csv').
        - Chuyển chuỗi các thuộc tính thành danh sách bằng args.Attribute.split(',').
    - Gọi hàm enhanced_radar_chart để vẽ biểu đồ:
        - Truyền dữ liệu, tên hai cầu thủ và danh sách thuộc tính vào hàm.

4.Cách sử dụng chương trình

- Chương trình được chạy từ dòng lệnh với các tham số bắt buộc:
    - python Cau3-2.py --p1 "Tên Cầu Thủ 1" --p2 "Tên Cầu Thủ 2" --Attribute "Thuộc tính 1,Thuộc tính 2,Thuộc tính 3"
    - **Ví dụ:**
        - python Cau3-2.py --p1 "Aaron Cresswell" --p2 "Aaron Ramsdale" --Attribute "Age,Matches Played,Starts,Minutes"



***Lưu ý rằng tên cầu thủ và các thuộc tính phải khớp với dữ liệu trong tệp results.csv.**