HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÁO CÁO BÀI TẬP LỚN 1

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn: Kim Ngọc Bách
Sinh viên: Đông Đức Nguyên
Mã sinh viên: B23DCVT311
Lớp: D23CQCEO6-B
Niên khóa: 2023 - 2028
Hệ đào tạo: Đại học chính quy

Hà Nội, 2025

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÁO CÁO BÀI TẬP LỚN 1

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn: Kim Ngọc Bách
Sinh viên: Đông Đức Nguyên
Mã sinh viên: B23DCVT311
Lớp: D23CQCEO6-B
Niên khóa: 2023 - 2028
Hệ đào tạo: Đại học chính quy

Hà Nội, 2025

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:**)

Hà Nội, ngày tháng năm 20...

Giảng viên

Mục lục

1 VẤN ĐỀ I **6**
- 1.1 Yêu cầu . . . 6
- 1.2 Các bước tiến hành . . . 7
- 1.3 Code thực tế và mô tả chi tiết . . . 8
  - 1.3.1 Hàm chính . . . 8
  - 1.3.2 Chi tiết các thao tác . . . 9
- 1.4 Kết quả và đánh giá . . . 16
  - 1.4.1 Kết quả: . . . 16
  - 1.4.2 Đánh giá: . . . 17

2 VẤN ĐỀ II **18**
- 2.1 Yêu cầu . . . 18
- 2.2 Các bước tiến hành . . . 18
- 2.3 Code thực tế và mô tả chi tiết . . . 19
  - 2.3.1 Hàm chính . . . 19
  - 2.3.2 Chi tiết các thao tác . . . 20
- 2.4 Kết quả và Đánh giá . . . 25
  - 2.4.1 Kết quả . . . 25
  - 2.4.2 Đánh giá . . . 35

3 VẤN ĐỀ III **36**
- 3.1 Yêu cầu . . . 36
- 3.2 Các bước tiến hành . . . 36
- 3.3 Code thực tế và mô tả chi tiết . . . 37
  - 3.3.1 Hàm chính . . . 37
  - 3.3.2 Chi tiết các thao tác . . . 38
- 3.4 Kết quả và đánh giá . . . 42
  - 3.4.1 Kết quả: . . . 42
  - 3.4.2 Đánh giá: . . . 46

4 VẤN ĐỀ IV **47**
- 4.1 Yêu cầu . . . 47
- 4.2 Các bước tiến hành . . . 47
  - 4.2.1 Yêu cầu 1 . . . 47
  - 4.2.2 Yêu cầu 2 . . . 47
- 4.3 Xử lý yêu cầu 1 . . . 47
  - 4.3.1 Code thực tế và mô tả chi tiết . . . 47
  - 4.3.2 Kết quả và Đánh giá . . . 50
- 4.4 Xử lý yêu cầu 2 . . . 52

4.4.1 Chọn model và feature để xử lý 52
4.4.2 Code thực tế và mô tả 56
4.4.3 Kiểm thử và đánh giá 58

Danh sách hình vẽ

1.1 Terminal sau khi chạy chương trình Problem1.py 16
1.2 File results.csv 17

2.1 Terminal sau khi chạy chương trình Problem2.py 26
2.2 Các tệp đầu ra sau khi chạy chương trình Problem2.py 26
2.3 File top_3.txt 27
2.4 File results2.csv 27
2.5 File Overall_League_Distribution_of_Player_Indexes.png 28
2.6 File Distribution_of_Performance_goals_by_Team.png 29
2.7 File Distribution_of_Performance_assists_by_Team.png 29
2.8 File Distribution_of_Expected_expected_goals_(xG)_by_Team.png . . 30
2.9 File Distribution_of_Tackles_Tkl_by_Team.png 30
2.10 File Distribution_of_Challenges_Att_by_Team.png 31
2.11 File Distribution_of_Blocks_Blocks_by_Team.png 31
2.12 File best_team_summary.txt 34

3.1 Terminal sau khi thực hiện chương tring Problem3.py 42
3.2 File: Elbow_Method_fo_Optimal_K.png 43
3.3 File: Silhouette_Score.png 44
3.4 File: PCA_of_Clusters_k=6.png 44

4.1 Terminal sau khi thực hiện function Task_1 50
4.2 File MoreThan900mins.csv 51
4.3 Terminal sau khi thực hiện function Task_2 59

MỞ ĐẦU

Trong kỷ nguyên số hóa, khả năng thu thập, xử lý và phân tích dữ liệu đã trở thành một kỹ năng thiết yếu trong nhiều lĩnh vực, bao gồm cả thể thao. Ngôn ngữ lập trình Python, với hệ sinh thái thư viện phong phú và mạnh mẽ như `Pandas`, `Scikit-learn`, `Selenium` và `BeautifulSoup`, cung cấp các công cụ hiệu quả để giải quyết các bài toán phân tích dữ liệu phức tạp. Báo cáo Bài tập lớn môn *Ngôn ngữ lập trình Python* này tập trung vào việc ứng dụng Python để thực hiện một loạt các nhiệm vụ liên quan đến phân tích dữ liệu bóng đá, cụ thể là từ giải Ngoại hạng Anh.

Mục tiêu chính của báo cáo bao gồm:

- Thu thập dữ liệu thống kê chi tiết của các cầu thủ từ các nguồn trực tuyến đáng tin cậy (`fbref.com`, `footballtransfers.com`) bằng kỹ thuật web scraping.
- Thực hiện các phân tích thống kê mô tả để khám phá dữ liệu, xác định các cầu thủ nổi bật và đặc điểm của các đội bóng.
- Áp dụng các thuật toán học máy không giám sát (K-means) kết hợp với kỹ thuật giảm chiều dữ liệu (PCA) để phân loại các cầu thủ thành các nhóm dựa trên đặc điểm thi đấu.
- Xây dựng mô hình học máy có giám sát (`Random Forest Regressor`) để ước tính giá trị chuyển nhượng của cầu thủ dựa trên các chỉ số thống kê và thông tin cá nhân.

Báo cáo được cấu trúc thành bốn phần chính (Vấn đề I, II, III, IV), mỗi phần giải quyết một yêu cầu cụ thể đã nêu trên. Thông qua việc thực hiện các nhiệm vụ này, báo cáo không chỉ minh họa khả năng ứng dụng của Python trong phân tích dữ liệu thể thao mà còn cung cấp những hiểu biết sâu sắc về hiệu suất cầu thủ và động lực thị trường chuyển nhượng.

Chương 1

VẤN ĐỀ I

1.1 Yêu cầu

- Collect statistical data [*] for all players who have played more than 90 minutes in the 2024-2025 English Premier League season.
- Data source: https://fbref.com/en/
- Save the result to a file named 'results.csv', where the result table has the following structure:
  - Each column corresponds to a statistic.
  - Players are sorted alphabetically by their first name.
  - Any statistic that is unavailable or inapplicable should be marked as "N/a".

* The required statistics are:
  - Nation
  - Team
  - Position
  - Age
  - Playing Time: matches played, starts, minutes
  - Performance: goals, assists, yellow cards, red cards
  - Expected: expected goals (xG), expected Assist Goals (xAG)
  - Progression: PrgC, PrgP, PrgR
  - Per 90 minutes: Gls, Ast, xG, xGA
  - Goalkeeping:
    * Performance: goals against per 90mins (GA90), Save%, CS%
    * Penalty Kicks: penalty kicks Save%
  - Shooting:
    * Standard: shots on target percentage (SoT%), Shot on Target per 90min (SoT/90), goals/shot (G/Sh), average shot distance (Dist)
  - Passing:

    - Total: passes completed (Cmp), pass completion (Cmp%), progressive passing distance (TotDist)
    - Short: pass completion (Cmp%)
    - Medium: pass completion (Cmp%)
    - Long: pass completion (Cmp%)
    - Expected: key passes (KP), pass into final third (1/3), pass into penalty area (PPA), CrsPA, PrgP
  - Goal and Shot Creation:
    - SCA: SCA, SCA90
    - GCA: GCA, GCA90
  - Defensive Actions:
    - Tackles: Tkl, TklW
    - Challenges: Att, Lost
    - Blocks: Blocks, Sh, Pass, Int
  - Possession:
    - Touches: Touches, Def Pen, Def 3rd, Mid 3rd, Att 3rd, Att Pen
    - Take-Ons: Att, Succ%, Tkld%
    - Carries: Carries, ProDist, ProgC, 1/3, CPA, Mis, Dis
    - Receiving: Rec, PrgR
  - Miscellaneous Stats:
    - Performance: Fls, Fld, Off, Crs, Recov
    - Aerial Duels: Won, Lost, Won%
  - Reference: `https://fbref.com/en/squads/822bd0ba/Liverpool-Stats`

1.2 Các bước tiến hành

1. Khởi tạo các trường dữ liệu:
   - Xác định các thông tin cần thu tập (các trường).
   - Tạo một dictionary (hoặc class) cầu thủ với giá trị mặc định 'N/a'.
   - Tạo một dictionary lưu tất cả các cầu thủ theo tên + đội bóng.
2. Lấy dữ liệu:
   - Lấy dữ liệu cơ bản ở trang tổng hợp (https://fbref.com/en/comps/9/stats/Premier-League-Stats) để khởi tạo đủ số lượng cầu thủ, lọc bỏ những cầu thủ không thi đấu trên 90 phút. Đưa vào dictionary để chuẩn bị cập nhật.
   - Cập nhật dữ liệu theo yêu cầu (chỉ cập nhật những cầu thủ có trong dictionary).
   - Cập nhât theo từng mục đã yêu cầu để đảm bảo đủ số lượng.
3. Xuất dữ liệu:
   - Xuất ra các trường theo đúng yêu cầu (chuyển sang list chứa nội dung theo đúng yêu cầu).
   - Cố định dữ liệu vào DataFrame
   - Xuất dữ liệu ra file result.csv

1.3 Code thực tế và mô tả chi tiết

1.3.1 Hàm chính

```
def Problem_1():

  print("Starting to retrieve basic data...")
  player_set_dict = create_Set_Players()
  print(f"Retrieved basic data for {len(player_set_dict)} players.")

  print("Updating goalkeeping data...")
  update_Set_Goalkeeping(player_set_dict)
  print("Updating shooting data...")
  update_Set_Shooting(player_set_dict)
  print("Updating passing data...")
  update_Set_Passing(player_set_dict)
  print("Updating goal and shot creation data...")
  update_Set_Goal_And_Shot_Creation_Data(player_set_dict)
  print("Updating defensive actions data...")
  update_Set_Defensive_Actions_Data(player_set_dict)
  print("Updating possession data...")
  update_Set_Possession(player_set_dict)
  print("Updating miscellaneous data...")
  update_Set_Miscellaneous_Data(player_set_dict)
  print("Data update completed.")

  export(player_set_dict)
  print("Program completed.")
```

Các thao tác tương tự như ở phần các bước tiến hành:
create_Set_Players() Dây là hàm khởi tạo dữ liệu đồng thời Lấy dữ liệu cơ bản ở trang tổng hợp để tạo ra một dictionary gồm toàn bộ các cầu thủ có thời gian thi đấu trên 90 phút. Lấy dữ liệu từ https://fbref.com/en/comps/9/stats/Premier-League-Stats

Các thao tác:

`update_Set_Goalkeeping(player_set_dict)`
`update_Set_Shooting(player_set_dict)`
`update_Set_Passing(player_set_dict)`
`update_Set_Goal_And_Shot_Creation_Data(player_set_dict)`
`update_Set_Defensive_Actions_Data(player_set_dict)`
`update_Set_Possession(player_set_dict)`
`update_Set_Miscellaneous_Data(player_set_dict)`

Đây là các thao tác trong quá trình cập nhật những dữ liệu theo yêu cầu:

- update_Set_Goalkeeping(player_set_dict): là thao tác đi cập nhật dữ liệu Goalkeeping, lấy dữ liệu từ: https://fbref.com/en/comps/9/keepers/Premier-League-Stats
- update_Set_Shooting(player_set_dict): là thao tác đi cập nhật dữ liệu Shooting, lấy dữ liệu từ: https://fbref.com/en/comps/9/shooting/Premier-League-Stats
- update_Set_Passing(player_set_dict): là thao tác đi cập nhật dữ liệu Passing, lấy dữ liệu từ: https://fbref.com/en/comps/9/passing/Premier-League-Stats
- update_Set_Goal_And_Shot_Creation_Data(player_set_dict): là thao tác đi cập nhật dữ liệu Goal And Shot Creation, lấy dữ liệu từ: https://fbref.com/en/comps/9/gca/Premie League-Stats

- update_Set_Defensive_Actions_Data(player_set_dict): là thao tác đi cập nhật dữ liệu Defensive Actions, lấy dữ liệu từ: https://fbref.com/en/comps/9/defense/Premier-League-Stats
- update_Set_Possession(player_set_dict): là thao tác đi cập nhật dữ liệu Possession, lấy dữ liệu từ: https://fbref.com/en/comps/9/possession/Premier-League-Stats
- update_Set_Miscellaneous_Data(player_set_dict): là thao tác đi cập nhật dữ liệu Miscellaneous, lấy dữ liệu từ: https://fbref.com/en/comps/9/misc/Premier-League-Stats

export(player_set_dict): là thao tác cố định dữ liệu và DataFrame rồi đưa ra file results.csv theo đúng yêu cầu

1.3.2 Chi tiết các thao tác

*Khởi tạo các dictionary chứa thông tin cầu thủ:

```
PLAYER_KEYS = ['name', 'nationality', 'position', 'team', 'age', 'games', 'games_starts',
     'minutes', 'goals', 'assist', 'cards_yellow', 'cards_red', 'xg', 'xg_assist', '
    progressive_carries', 'progressive_passes', 'progressive_passes_received', '
    goals_per90', 'assists_per90', 'xg_per90', 'xg_assist_per90', 'gk_goals_against_per90
    ', 'gk_save_pct', 'gk_clean_sheets_pct', 'gk_pens_save_pct', 'shots_on_target_pct', '
    shots_on_target_per90', 'goals_per_shot', 'average_shot_distance', 'passes_completed'
    , 'passes_pct', 'passes_total_distance', 'passes_pct_short', 'passes_pct_medium', '
    passes_pct_long', 'assisted_shots', 'passes_into_final_third', '
    passes_into_penalty_area', 'crosses_into_penalty_area', 'sca', 'sca_per90', 'gca', '
    gca_per90', 'tackles', 'tackles_won', 'challenges', 'challenges_lost', 'blocks', '
    blocked_shots', 'blocked_passes', 'interceptions', 'touches', 'touches_def_pen_area',
     'touches_def_3rd', 'touches_mid_3rd', 'touches_att_3rd', 'touches_att_pen_area', '
    take_ons', 'take_ons_won_pct', 'take_ons_tackled_pct', 'carries', '
    carries_progressive_distance', 'carries_into_final_third', 'carries_into_penalty_area
    ', 'miscontrols', 'dispossessed', 'passes_received', 'fouls', 'fouled', 'offsides', '
    crosses', 'ball_recoveries', 'aerials_won', 'aerials_lost', 'aerials_won_pct']

def create_default_player_dict(): return {key: 'N/a' for key in PLAYER_KEYS}
```

Thao tác này nhằm mục đích tạo ra một dictionary cầu thủ với các thuộc tính ở trong danh sách PLAYER_KEYS đã được khai báo trước đó, ta đặt giá trị ban đầu cho tất cả các dữ liệu là 'N/a' để những giá trị nào không tìm thấy (unavailable or inapplicable) sẽ được mặc định mang giá trị này.

*create_Set_Players() - Tạo dictionary cầu thủ:

```
def create_Set_Players():
    driver = webdriver.Chrome()
    url = 'https://fbref.com/en/comps/9/stats/Premier-League-Stats'
    driver.get(url)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    x = soup.find('table', attrs={'id': 'stats_standard'})
    cnt = 0
    player_set = {}
    for i in range(589):
        cnt += 1
        if cnt == 26:
            cnt = 0
            continue
        table = x.find('tr', attrs={'data-row': f'{str(i)}'})
        if not table:
            continue
        player_data = create_default_player_dict()
        player_data['name'] = table.find('td', attrs={'data-stat': 'player'}).text.strip
            ()
```

```
20          player_data['nationality'] = table.find('td', attrs={'data-stat': 'nationality'})
                .text.strip()
21          player_data['position'] = table.find('td', attrs={'data-stat': 'position'}).text.
                strip()
22          player_data['team'] = table.find('td', attrs={'data-stat': 'team'}).text.strip()
23          player_data['age'] = table.find('td', attrs={'data-stat': 'age'}).text.strip()
24          player_data['games'] = table.find('td', attrs={'data-stat': 'games'}).text.strip
                ()
25          player_data['games_starts'] = table.find('td', attrs={'data-stat': 'games_starts'
                }).text.strip()
26          minutes_str = table.find('td', attrs={'data-stat': 'minutes'}).text.strip()
27          player_data['minutes'] = minutes_str
28          minutes_str_cleaned = minutes_str.replace(',', '')
29          if int(minutes_str_cleaned) <= 90:
30              continue
31          player_data['goals'] = table.find('td', attrs={'data-stat': 'goals'}).text.strip
                ()
32          player_data['assist'] = table.find('td', attrs={'data-stat': 'assists'}).text.
                strip()
33          player_data['cards_yellow'] = table.find('td', attrs={'data-stat': 'cards_yellow'
                }).text.strip()
34          player_data['cards_red'] = table.find('td', attrs={'data-stat': 'cards_red'}).
                text.strip()
35          player_data['xg'] = table.find('td', attrs={'data-stat': 'xg'}).text.strip()
36          player_data['xg_assist'] = table.find('td', attrs={'data-stat': 'xg_assist'}).
                text.strip()
37          player_data['progressive_carries'] = table.find('td', attrs={'data-stat': '
                progressive_carries'}).text.strip()
38          player_data['progressive_passes'] = table.find('td', attrs={'data-stat': '
                progressive_passes'}).text.strip()
39          player_data['progressive_passes_received'] = table.find('td', attrs={'data-stat':
                 'progressive_passes_received'}).text.strip()
40          player_data['goals_per90'] = table.find('td', attrs={'data-stat': 'goals_per90'})
                .text.strip()
41          player_data['assists_per90'] = table.find('td', attrs={'data-stat': '
                assists_per90'}).text.strip()
42          player_data['xg_per90'] = table.find('td', attrs={'data-stat': 'xg_per90'}).text.
                strip()
43          player_data['xg_assist_per90'] = table.find('td', attrs={'data-stat': '
                xg_assist_per90'}).text.strip()
44          player_key = str(player_data['name']) + str(player_data['team'])
45          player_set[player_key] = player_data
46      driver.quit()
47      return player_set
```

Để tránh bị chặn và đảm bảo đầy đủ, chính xác và không bị thất bại trong quá trình lấy dữ liệu từ trang web sử dụng webdriver lấy từ thư viện selenium đưa nó vào biến driver và tiến hành lấy dữ liệu [dòng 2 - 5]. Khi đã có dữ liệu của trang web, chuẩn hóa bằng BeautifulSoup [dòng 6]. Kết thúc quá trình lấy dữ liệu từ trang web.
Tiến hành quá trình tìm kiếm, đầu tiên để giảm thời gian tìm kiếm ta chỉ lấy dữ liệu cần thiết (bảng thông tin vào trong biến x) [dòng 7]. Ta tạo một biến cao để tránh những dòng mang thông tin tiêu đề [dòng 8, 11 - 14]. Tạo một tập dữ liệu cầu thủ để lưu các cầu thủ hợp lệ [dòng 9].
Thực hiện lặp để lấy toàn bộ thông tin cầu thủ. Với mỗi vòng lặp ta tạo ra một dictionary chứa thông tin cầu thủ mới [dòng 18]. Ta tìm lần lượt tên, quốc tịch, đội bóng, tuổi, số trận thi đấu, số trận ở vị trí bắt đầu, thời gian thi đấu rồi gán vào các key trong dictionary [dòng 19 - 27]. Ta chuẩn hóa lại thời gian thi đấu rồi so sánh nếu trên 90 tiếp tục lấy các dữ liệu còn lại, nếu không chuyển đến cầu thủ tiếp theo. Tiếp tục lấy toàn bộ dữ liệu đến khi hết bảng ta gán dictionary vừa tạo vào dictionary chứa toàn những cầu thủ hợp lệ tạo ở dòng 9, đặt key là tên cầu thủ + tên đội.
Khi kết thúc lặp ta quit driver và trả về dictionary chứa toàn những cầu thủ hợp lệ tạo ở dòng 9.

*Các hàm cập nhật:

```
1 def update_Set_Goalkeeping(player_set):
```

```
    driver = webdriver.Chrome()
    url = 'https://fbref.com/en/comps/9/keepers/Premier-League-Stats'
    driver.get(url)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    x = soup.find('table', attrs={'id': 'stats_keeper'})
    for i in range (43):
        if i == 25: continue
        table = x.find('tr', attrs={'data-row':f'{str(i)}'})
        if not table: continue

        name = table.find('td', attrs={'data-stat':'player'}).text.strip()
        team = table.find('td', attrs={'data-stat':'team'}).text.strip()
        player_key = str(name) + str(team)

        if player_key in player_set:
            player_set[player_key]['gk_goals_against_per90'] = table.find('td', attrs={'
                data-stat':'gk_goals_against_per90'}).text.strip()
            player_set[player_key]['gk_save_pct'] = table.find('td', attrs={'data-stat':'
                gk_save_pct'}).text.strip()
            player_set[player_key]['gk_clean_sheets_pct'] = table.find('td', attrs={'data
                -stat':'gk_clean_sheets_pct'}).text.strip()
            player_set[player_key]['gk_pens_save_pct'] = table.find('td', attrs={'data-
                stat':'gk_pens_save_pct'}).text.strip()
    driver.quit()

def update_Set_Shooting(player_set):
    driver = webdriver.Chrome()
    url = 'https://fbref.com/en/comps/9/shooting/Premier-League-Stats'
    driver.get(url)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    x = soup.find('table', attrs={'id': 'stats_shooting'})
    cnt = 0
    for i in range(589):
        cnt+=1
        if cnt==26:
            cnt=0
            continue
        table = x.find('tr', attrs={'data-row':f'{str(i)}'})
        if not table: continue

        name = table.find('td', attrs={'data-stat':'player'}).text.strip()
        team = table.find('td', attrs={'data-stat':'team'}).text.strip()
        player_key = str(name) + str(team)

        if player_key in player_set:
            player_set[player_key]['shots_on_target_pct'] = table.find('td', attrs={'data
                -stat':'shots_on_target_pct'}).text.strip()
            player_set[player_key]['shots_on_target_per90'] = table.find('td', attrs={'
                data-stat':'shots_on_target_per90'}).text.strip()
            player_set[player_key]['goals_per_shot'] = table.find('td', attrs={'data-stat
                ':'goals_per_shot'}).text.strip()
            player_set[player_key]['average_shot_distance'] = table.find('td', attrs={'
                data-stat':'average_shot_distance'}).text.strip()
    driver.quit()

def update_Set_Passing(player_set):
    driver = webdriver.Chrome()
    url = 'https://fbref.com/en/comps/9/passing/Premier-League-Stats'
    driver.get(url)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    x = soup.find('table', attrs={'id': 'stats_passing'})
    cnt = 0
    for i in range(589):
        cnt+=1
        if cnt==26:
            cnt=0
            continue
        table = x.find('tr', attrs={'data-row':f'{str(i)}'})
        if not table: continue

        name = table.find('td', attrs={'data-stat':'player'}).text.strip()
        team = table.find('td', attrs={'data-stat':'team'}).text.strip()
```

```
69          player_key = str(name) + str(team)
70
71          if player_key in player_set:
72              player_set[player_key]['passes_completed'] = table.find('td', attrs={'data-
                    stat':'passes_completed'}).text.strip()
73              player_set[player_key]['passes_pct'] = table.find('td', attrs={'data-stat':'
                    passes_pct'}).text.strip()
74              player_set[player_key]['passes_total_distance'] = table.find('td', attrs={'
                    data-stat':'passes_total_distance'}).text.strip()
75              player_set[player_key]['passes_pct_short'] = table.find('td', attrs={'data-
                    stat':'passes_pct_short'}).text.strip()
76              player_set[player_key]['passes_pct_medium'] = table.find('td', attrs={'data-
                    stat':'passes_pct_medium'}).text.strip()
77              player_set[player_key]['passes_pct_long'] = table.find('td', attrs={'data-
                    stat':'passes_pct_long'}).text.strip()
78              player_set[player_key]['assisted_shots'] = table.find('td', attrs={'data-stat
                    ':'assisted_shots'}).text.strip()
79              player_set[player_key]['passes_into_final_third'] = table.find('td', attrs={'
                    data-stat':'passes_into_final_third'}).text.strip()
80              player_set[player_key]['passes_into_penalty_area'] = table.find('td', attrs={
                    'data-stat':'passes_into_penalty_area'}).text.strip()
81              player_set[player_key]['crosses_into_penalty_area'] = table.find('td', attrs
                    ={'data-stat':'crosses_into_penalty_area'}).text.strip()
82              player_set[player_key]['progressive_passes'] = table.find('td', attrs={'data-
                    stat':'progressive_passes'}).text.strip()
83      driver.quit()
84
85
86  def update_Set_Goal_And_Shot_Creation_Data(player_set):
87      driver = webdriver.Chrome()
88      url = 'https://fbref.com/en/comps/9/gca/Premier-League-Stats'
89      driver.get(url)
90      page_source = driver.page_source
91      soup = BeautifulSoup(page_source, 'html.parser')
92      x = soup.find('table', attrs={'id': 'stats_gca'})
93      cnt = 0
94      for i in range(589):
95          cnt+=1
96          if cnt==26:
97              cnt=0
98              continue
99          table = x.find('tr', attrs={'data-row':f'{str(i)}'})
100         if not table: continue
101
102         name = table.find('td', attrs={'data-stat':'player'}).text.strip()
103         team = table.find('td', attrs={'data-stat':'team'}).text.strip()
104         player_key = str(name) + str(team)
105
106         if player_key in player_set:
107             player_set[player_key]['sca'] = table.find('td', attrs={'data-stat':'sca'}).
                    text.strip()
108             player_set[player_key]['sca_per90'] = table.find('td', attrs={'data-stat':'
                    sca_per90'}).text.strip()
109             player_set[player_key]['gca'] = table.find('td', attrs={'data-stat':'gca'}).
                    text.strip()
110             player_set[player_key]['gca_per90'] = table.find('td', attrs={'data-stat':'
                    gca_per90'}).text.strip()
111     driver.quit()
112
113 def update_Set_Defensive_Actions_Data(player_set):
114     driver = webdriver.Chrome()
115     url = 'https://fbref.com/en/comps/9/defense/Premier-League-Stats'
116     driver.get(url)
117     page_source = driver.page_source
118     soup = BeautifulSoup(page_source, 'html.parser')
119     x = soup.find('table', attrs={'id': 'stats_defense'})
120     cnt = 0
121     for i in range(589):
122         cnt+=1
123         if cnt==26:
124             cnt=0
125             continue
126         table = x.find('tr', attrs={'data-row':f'{str(i)}'})
127         if not table: continue
128
```

```
129         name = table.find('td', attrs={'data-stat':'player'}).text.strip()
130         team = table.find('td', attrs={'data-stat':'team'}).text.strip()
131         player_key = str(name) + str(team)
132 
133         if player_key in player_set:
134             player_set[player_key]['tackles'] = table.find('td', attrs={'data-stat':'
                    tackles'}).text.strip()
135             player_set[player_key]['tackles_won'] = table.find('td', attrs={'data-stat':'
                    tackles_won'}).text.strip()
136             player_set[player_key]['challenges'] = table.find('td', attrs={'data-stat':'
                    challenges'}).text.strip()
137             player_set[player_key]['challenges_lost'] = table.find('td', attrs={'data-
                    stat':'challenges_lost'}).text.strip()
138             player_set[player_key]['blocks'] = table.find('td', attrs={'data-stat':'
                    blocks'}).text.strip()
139             player_set[player_key]['blocked_shots'] = table.find('td', attrs={'data-stat'
                    :'blocked_shots'}).text.strip()
140             player_set[player_key]['blocked_passes'] = table.find('td', attrs={'data-stat
                    ':'blocked_passes'}).text.strip()
141             player_set[player_key]['interceptions'] = table.find('td', attrs={'data-stat'
                    :'interceptions'}).text.strip()
142     driver.quit()
143 
144 def update_Set_Possession(player_set):
145     driver = webdriver.Chrome()
146     url = 'https://fbref.com/en/comps/9/possession/Premier-League-Stats'
147     driver.get(url)
148     page_source = driver.page_source
149     soup = BeautifulSoup(page_source, 'html.parser')
150     x = soup.find('table', attrs={'id': 'stats_possession'})
151     cnt = 0
152     for i in range(589):
153         cnt+=1
154         if cnt==26:
155             cnt=0
156             continue
157         table = x.find('tr', attrs={'data-row':f'{str(i)}'})
158         if not table: continue
159 
160         name = table.find('td', attrs={'data-stat':'player'}).text.strip()
161         team = table.find('td', attrs={'data-stat':'team'}).text.strip()
162         player_key = str(name) + str(team)
163 
164         if player_key in player_set:
165             player_set[player_key]['touches'] = table.find('td', attrs={'data-stat':'
                    touches'}).text.strip()
166             player_set[player_key]['touches_def_pen_area'] = table.find('td', attrs={'
                    data-stat':'touches_def_pen_area'}).text.strip()
167             player_set[player_key]['touches_def_3rd'] = table.find('td', attrs={'data-
                    stat':'touches_def_3rd'}).text.strip()
168             player_set[player_key]['touches_mid_3rd'] = table.find('td', attrs={'data-
                    stat':'touches_mid_3rd'}).text.strip()
169             player_set[player_key]['touches_att_3rd'] = table.find('td', attrs={'data-
                    stat':'touches_att_3rd'}).text.strip()
170             player_set[player_key]['touches_att_pen_area'] = table.find('td', attrs={'
                    data-stat':'touches_att_pen_area'}).text.strip()
171             player_set[player_key]['take_ons'] = table.find('td', attrs={'data-stat':'
                    take_ons'}).text.strip()
172             player_set[player_key]['take_ons_won_pct'] = table.find('td', attrs={'data-
                    stat':'take_ons_won_pct'}).text.strip()
173             player_set[player_key]['take_ons_tackled_pct'] = table.find('td', attrs={'
                    data-stat':'take_ons_tackled_pct'}).text.strip()
174             player_set[player_key]['carries'] = table.find('td', attrs={'data-stat':'
                    carries'}).text.strip()
175             player_set[player_key]['carries_progressive_distance'] = table.find('td',
                    attrs={'data-stat':'carries_progressive_distance'}).text.strip()
176             player_set[player_key]['carries_into_final_third'] = table.find('td', attrs={
                    'data-stat':'carries_into_final_third'}).text.strip()
177             player_set[player_key]['carries_into_penalty_area'] = table.find('td', attrs
                    ={'data-stat':'carries_into_penalty_area'}).text.strip()
178             player_set[player_key]['miscontrols'] = table.find('td', attrs={'data-stat':'
                    miscontrols'}).text.strip()
179             player_set[player_key]['dispossessed'] = table.find('td', attrs={'data-stat':
                    'dispossessed'}).text.strip()
180             player_set[player_key]['passes_received'] = table.find('td', attrs={'data-
```

```
                stat':'passes_received'}).text.strip()
    driver.quit()


def update_Set_Miscellaneous_Data(player_set):
    driver = webdriver.Chrome()
    url = 'https://fbref.com/en/comps/9/misc/Premier-League-Stats'
    driver.get(url)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    x = soup.find('table', attrs={'id': 'stats_misc'})
    cnt = 0
    for i in range(589):
        cnt+=1
        if cnt==26:
            cnt=0
            continue
        table = x.find('tr', attrs={'data-row':f'{str(i)}'})
        if not table: continue

        name = table.find('td', attrs={'data-stat':'player'}).text.strip()
        team = table.find('td', attrs={'data-stat':'team'}).text.strip()
        player_key = str(name) + str(team)

        if player_key in player_set:
            player_set[player_key]['fouls'] = table.find('td', attrs={'data-stat':'fouls'
                }).text.strip()
            player_set[player_key]['fouled'] = table.find('td', attrs={'data-stat':'
                fouled'}).text.strip()
            player_set[player_key]['offsides'] = table.find('td', attrs={'data-stat':'
                offsides'}).text.strip()
            player_set[player_key]['crosses'] = table.find('td', attrs={'data-stat':'
                crosses'}).text.strip()
            player_set[player_key]['ball_recoveries'] = table.find('td', attrs={'data-
                stat':'ball_recoveries'}).text.strip()
            player_set[player_key]['aerials_won'] = table.find('td', attrs={'data-stat':'
                aerials_won'}).text.strip()
            player_set[player_key]['aerials_lost'] = table.find('td', attrs={'data-stat':
                'aerials_lost'}).text.strip()
            player_set[player_key]['aerials_won_pct'] = table.find('td', attrs={'data-
                stat':'aerials_won_pct'}).text.strip()
    driver.quit()
```

Các hàm cập nhật thao tác tương tự hoàn toàn về cách lấy dữ liệu từ web và lấy thông tin chỉ khác là sẽ không đi tạo dictionary mới nữa mà truyền vào các dictionary sau khi lấy được tên và đội bóng sẽ đi kiểm tra xem có tồn tại cầu thủ đang xét trong dictionary không nếu có sẽ tiến hành cập nhật những dữ liệu theo đúng yêu cầu nếu không thì bỏ qua chuyển sang cầu thủ tiếp theo.

*Thao tác xuất dữ liệu

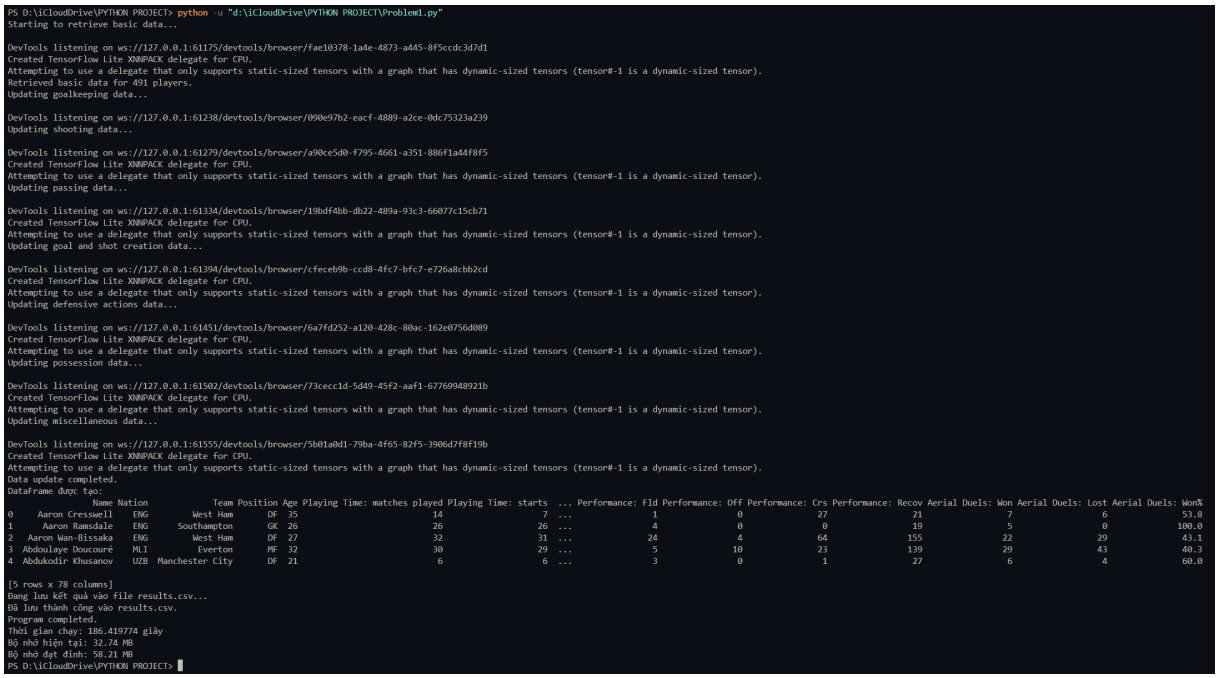
```
def export_player_data(player_dict):
    export_order_keys = ['name', 'nationality', 'team', 'position', 'age', 'games', '
        games_starts', 'minutes', 'goals', 'assist', 'cards_yellow', 'cards_red', 'xg', '
        xg_assist', 'progressive_carries', 'progressive_passes', '
        progressive_passes_received', 'goals_per90', 'assists_per90', 'xg_per90', '
        xg_assist_per90', 'gk_goals_against_per90', 'gk_save_pct', 'gk_clean_sheets_pct',
         'gk_pens_save_pct', 'shots_on_target_pct', 'shots_on_target_per90', '
        goals_per_shot', 'average_shot_distance', 'passes_completed', 'passes_pct', '
        passes_total_distance', 'passes_pct_short', 'passes_pct_medium', 'passes_pct_long
        ', 'assisted_shots', 'passes_into_final_third', 'passes_into_penalty_area', '
        crosses_into_penalty_area', 'progressive_passes', 'sca', 'sca_per90', 'gca', '
        gca_per90', 'tackles', 'tackles_won', 'challenges', 'challenges_lost', 'blocks',
        'blocked_shots', 'blocked_passes', 'interceptions', 'touches', '
        touches_def_pen_area', 'touches_def_3rd', 'touches_mid_3rd', 'touches_att_3rd', '
        touches_att_pen_area', 'take_ons', 'take_ons_won_pct', 'take_ons_tackled_pct', '
        carries', 'carries_progressive_distance', 'progressive_carries', '
        carries_into_final_third',   'carries_into_penalty_area', 'miscontrols', '
        dispossessed', 'passes_received', 'progressive_passes_received', 'fouls', 'fouled
        ', 'offsides', 'crosses', 'ball_recoveries', 'aerials_won', 'aerials_lost', '
        aerials_won_pct']
    nationality = player_dict.get('nationality', 'N/a')
```

```
4       age = player_dict.get('age', 'N/a')
5       nationality_processed = nationality.split()[1] if ' ' in nationality else nationality
6       age_processed = age.split('-')[0] if '-' in age else age
7
8       exported_list = []
9       for key in export_order_keys:
10          if key == 'nationality':
11              exported_list.append(nationality_processed)
12          elif key == 'age':
13               exported_list.append(age_processed)
14          else:
15              exported_list.append(player_dict.get(key, 'N/a'))
16
17      return exported_list
18
19
20  def get_player_name_from_dict(player_dict):
21      return player_dict.get('name', '')
22
23  def export(player_set_dict):
24      playerlist = list(player_set_dict.values())
25      playerlist.sort(key=get_player_name_from_dict)
26      result = []
27      for player_dict in playerlist:
28          result.append(export_player_data(player_dict))
29      column_names = ['Name', 'Nation', 'Team', 'Position', 'Age', 'Playing Time: matches
            played', 'Playing Time: starts', 'Playing Time: minutes', 'Performance: goals', '
            Performance: assists', 'Performance: yellow cards', 'Performance: red cards', '
            Expected: expected goals (xG)', 'Expected: expedted Assist Goals (xAG)', '
            Progression: PrgC', 'Progression: PrgP', 'Progression: PrgR', 'Per 90 minutes:
            Gls', 'Per 90 minutes: Ast', 'Per 90 minutes: xG', 'Per 90 minutes: xGA', '
            Performance: goals against per 90mins (GA90)', 'Performance: Save%', 'Performance
            : CS%', 'Penalty Kicks: penalty kicks Save%', 'Standard: shoots on target
            percentage (SoT%)', 'Standard: Shoot on Target per 90min (SoT/90)', 'Standard:
            goals/shot (G/sh)', 'Standard: average shoot distance (Dist)', 'Total: passes
            completed (Cmp)', 'Total: Pass completion (Cmp%)', 'Total: progressive passing
            distance (TotDist)', 'Short: Pass completion (Cmp%)', 'Medium: Pass completion (
            Cmp%)', 'Long: Pass completion (Cmp%)', 'Expected: key passes (KP)', 'Expected:
            pass into final third (1/3)', 'Expected: pass into penalty area (PPA)', 'Expected
            : CrsPA', 'Expected: PrgP', 'SCA: SCA', 'SCA: SCA90', 'GCA: GCA', 'GCA: GCA90', '
            Tackles: Tkl', 'Tackles: TklW', 'Challenges: Att', 'Challenges: Lost', 'Blocks:
            Blocks', 'Blocks: Sh', 'Blocks: Pass', 'Blocks: Int', 'Touches: Touches', '
            Touches: Def Pen', 'Touches: Def 3rd', 'Touches: Mid 3rd', 'Touches: Att 3rd', '
            Touches: Att Pen', 'Take-Ons: Att', 'Take-Ons: Succ%', 'Take-Ons: Tkld%', '
            Carries: Carries', 'Carries: ProDist', 'Carries: ProgC', 'Carries: 1/3', 'Carries
            : CPA', 'Carries: Mis', 'Carries: Dis', 'Receiving: Rec', 'Receiving: PrgR', '
            Performance: Fls', 'Performance: Fld', 'Performance: Off', 'Performance: Crs', '
            Performance: Recov', 'Aerial Duels: Won', 'Aerial Duels: Lost', 'Aerial Duels:
            Won%']
30
31      dataFrame = pd.DataFrame(result , columns=column_names)
32
33      output_filename = 'results.csv'
34      dataFrame.to_csv(output_filename, index=False, encoding='utf-8-sig')
```

Ở thao tác này đầu tiên là đưa toàn bộ dữ liệu lấy được ở dictionary về dạng list sau đó thực hiện sắp xếp theo tên cầu thủ đúng như yêu cầu, dùng 1 hàm hỗ trợ việc lấy tên ở [dòng 20]. Lúc này trong list vẫn là các dictionary chứa thông tin cầu thủ nhưng đã được sắp xếp, chạy một vòng lặp để đưa toàn bộ dữ liệu chuẩn đầu ra vào list result. Với mỗi lần lặp, ta chuyển dữ liệu về đúng đầu ra bằng hàm export_player_data() [dòng 1]. Sau khi hoàn tất việc chuyển đổi, cố định dữ liệu và DataFrame với các trường như yêu cầu [dòng 31]. Xuất dữ liệu sang file results.csv rồi kết thúc.

1.4 Kết quả và đánh giá

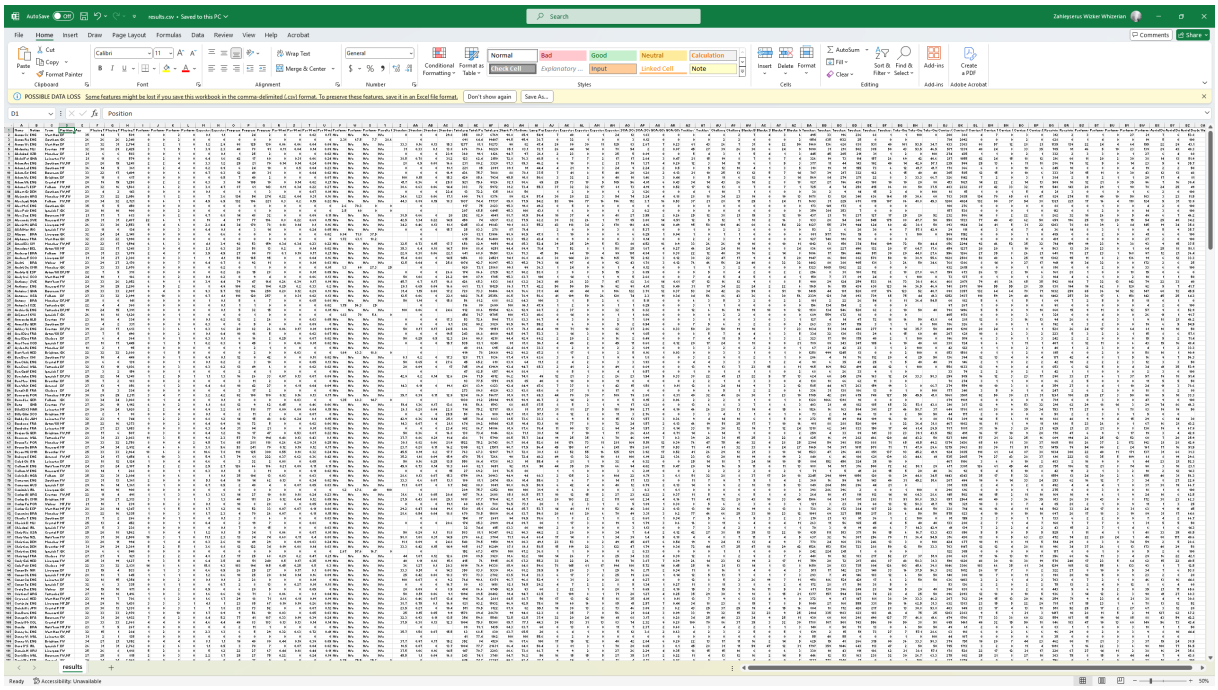
1.4.1 Kết quả:



Hình 1.1: Terminal sau khi chạy chương trình Problem1.py

Thu thập Dữ liệu: Script bắt đầu bằng việc thông báo "Starting to retrieve team data..."(Bắt đầu truy xuất dữ liệu đội) và sau đó là "Retrieved basic data for 491 players."(Đã truy xuất dữ liệu cơ bản cho 419 cầu thủ). Tiếp theo, nó cập nhật các loại dữ liệu khác nhau như dữ liệu hiệu suất, dữ liệu đội hình xuất phát, dữ liệu tạo cơ hội ghi bàn và sút bóng, dữ liệu hành động phòng ngự, và dữ liệu kiểm soát bóng.
Kết nối DevTools: Các dòng "DevTools listening on ws://127.0.0.1:..."lặp lại nhiều lần. Điều này thường xuất hiện khi một công cụ tự động hóa trình duyệt (Selenium) được sử dụng, có thể là để lấy dữ liệu từ các trang web.
Hiển thị Dữ liệu Dạng Bảng: Sau khi cập nhật dữ liệu xong ("Data update completed."), script hiển thị một phần của bảng dữ liệu. Bảng hiển thị 5 dòng đầu tiên và có tổng cộng 78 cột.
Lưu và Xếp hạng Kết quả: Script thông báo "All data written to the file results.csv."(Tất cả dữ liệu đã được ghi vào tệp results.csv) và sau đó "Attempting to rank the results.csv."(Đang cố gắng xếp hạng tệp results.csv).
Hoàn thành và Thời gian Thực thi: Cuối cùng, script báo "Process complete."(Quá trình hoàn tất) và cho biết thời gian thực thi là khoảng 186.419472 giây; Bộ nhớ tối đa sử dụng: 32.74 MB; Bộ nhớ đạt đỉnh: 58.21 MB.

Tệp dữ liệu results.csv tập hợp thông tin thống kê chi tiết của 491 vận động viên bóng đá chuyên nghiệp, với mỗi đối tượng được mô tả thông qua 78 biến số định lượng và định tính. Các trường dữ liệu bao gồm thông tin cá nhân (họ tên, quốc tịch, câu lạc bộ chủ quản, vị trí thi đấu, độ tuổi), thông tin về thời lượng thi đấu (số trận ra sân, số trận đá chính, tổng số phút thi đấu), thành tích thi đấu (số bàn thắng, số đường kiến tạo, số thẻ phạt), cùng các chỉ số dự đoán hiệu suất (Expected Goals - xG, Expected Assisted Goals



Hình 1.2: File results.csv

- xAG). Ngoài ra, bộ dữ liệu còn cung cấp các chỉ số liên quan đến kỹ năng kiểm soát bóng, chuyền bóng, rê dắt, phòng ngự, khả năng tiến bộ bóng (progression), hiệu suất trung bình chuẩn hóa trên 90 phút thi đấu, cũng như hiệu quả tranh chấp tay đôi trên không.

1.4.2 Đánh giá:

Chương trình trong tập tin Problem1.py triển khai quy trình thu thập, tổng hợp và xuất dữ liệu thống kê chi tiết của các cầu thủ Premier League thông qua kỹ thuật web scraping, kết hợp Selenium và BeautifulSoup. Dữ liệu được trích xuất từ nhiều bảng thống kê chuyên biệt trên fbref.com, bao gồm các chỉ số tiêu chuẩn, chỉ số thủ môn, khả năng dứt điểm, chuyền bóng, kiểm soát bóng, phòng ngự và các chỉ số hỗn hợp khác. Mỗi cầu thủ được biểu diễn dưới dạng một từ điển với khóa định danh duy nhất (kết hợp tên và câu lạc bộ), đảm bảo tính toàn vẹn dữ liệu trong quá trình cập nhật từ nhiều nguồn. Sau khi thu thập đầy đủ, chương trình xây dựng một DataFrame gồm 78 thuộc tính và xuất kết quả ra tệp CSV định dạng chuẩn, phục vụ cho phân tích dữ liệu thống kê sâu hơn.

Về mặt thời gian thực thi, chương trình hoàn thành toàn bộ quy trình trong khoảng 186.4 giây, thể hiện khả năng thu thập dữ liệu chi tiết và chính xác nhưng hiệu suất còn thấp. Nguyên nhân chủ yếu đến từ việc khởi tạo nhiều phiên trình duyệt Chrome độc lập cho từng loại bảng dữ liệu, gây ra chi phí khởi động, tải trang và đóng trình duyệt lặp lại nhiều lần. Về mặt sử dụng bộ nhớ, chương trình hoạt động hiệu quả, chỉ sử dụng tối đa khoảng 32.74 MB bộ nhớ RAM trong suốt quá trình thu thập và xử lý, cho thấy sự tiết kiệm tài nguyên bộ nhớ nhờ vào việc lưu trữ dữ liệu theo cấu trúc đơn giản và xử lý tuần tự.

Tóm lại, chương trình đạt độ chính xác cao và tổ chức dữ liệu hệ thống tốt, song tồn tại nhược điểm rõ rệt về thời gian thực thi do thiết kế chưa tối ưu hóa quy trình truy cập web, trong khi ưu điểm nổi bật là mức sử dụng bộ nhớ thấp, phù hợp với các hệ thống hạn chế về tài nguyên.

Chương 2

VẤN ĐỀ II

2.1 Yêu cầu

- Identify the top 3 players with the highest and lowest scores for each statistic. Save result to a file named `top_3.txt`.
- Find the median for each statistic. Calculate the mean and standard deviation for each statistic across all players and for each team. Save the results to a file named 'results2.csv' with the following format:

		Median of Attribute 1	Mean of Attribute 1	Std of Attribute 1	...
0	all				
1	Team 1				
⋮	⋮				
n	Team n				

Bảng 2.1: Statistics per team

- Plot a histogram showing the distribution of each statistic for all players in the league and each team.
- Identify the team with the highest scores for each statistic. Based on your analysis, which team do you think is performing the best in the 2024-2025 Premier League season?

2.2 Các bước tiến hành

1. Thực hiện phân tích dữ liệu thống kê của các cầu thủ bóng đá từ file CSV đầu vào (results.csv). Đầu tiên, đọc dữ liệu, xử lý các cột số liệu và phân chia thành nhóm thông tin định tính và định lượng.
2. Sau đó, chương trình tiến hành tìm kiếm Top 3 cầu thủ có chỉ số cao nhất và thấp nhất cho từng loại thống kê. Đưa kết quả ra file top3.txt như yêu cầu.
3. Tính toán các thống kê tổng hợp như trung bình, trung vị và độ lệch chuẩn cho toàn bộ giải đấu cũng như cho từng đội bóng riêng biệt. Các kết quả này được lưu lại thành file results2.csv như yêu cầu đề bài.

4. Trực quan hóa dữ liệu bằng cách vẽ các biểu đồ phân phối cho các chỉ số, bao gồm cả biểu đồ tổng thể toàn giải và biểu đồ chi tiết cho từng đội bóng. Các biểu đồ này được lưu dưới dạng hình ảnh.

5. Xác định đội bóng mạnh nhất dựa trên việc so sánh trung bình các chỉ số, và tổng hợp đội có nhiều chỉ số tốt nhất thành "Best Overall Team". Kết quả này cũng được ghi lại.

2.3 Code thực tế và mô tả chi tiết

2.3.1 Hàm chính

```
def Problem_2():
    df, stats_cols, non_numeric_cols = read_data()
    # 1. Find top 3 highest and lowest for each statistic
    Find_Top_3(df, stats_cols)
    # 2. Calculate Median, Mean, Std for each statistic
    Calculate_For_Each_Statistic(df, stats_cols)
    # 3. Plotting
    Plotting(df)
    # 4. Identify the best team for each statistic
    Best_Team_Summary(df, stats_cols)
```

Các thao tác được thực hiện theo đúng trình tự đã mô tả ở phần các bước tiến hành:

* Hàm read_data(): Nhập dữ liệu đầu vào:

- Nhập dữ liệu đầu vào từ file results.csv đã lưu ở Vấn đề 1 (thay vì đi lấy dữ liệu 1 lần nữa).
- Trả về DataFrame, Cột thống kê, cột không chứa số (dữ liệu phân tích).

* Hàm Find_Top_3(df, stats_cols)

- Nạp dữ liệu vào là DataFrame và các cột thống kê (Kết quả của fuction read_data()).
- Thực hiện việc tìm top 3 (cao nhất và thấp nhất của mỗi chỉ số).
- Xuất dữ liệu ra file top_3.txt theo yêu cầu.

* Calculate_For_Each_Statistic(df, stats_cols)

- Nạp dữ liệu vào là DataFrame và các cột thống kê (Kết quả của fuction read_data()).
- Tiến hành thực hiện các bước để đưa các giá các giá trị ở mỗi team (trung bình, độ lệch) cho tất cả các trường.
- Đưa về đúng các trường như yêu cầu rồi cố định vào trong DataFrame.
- Xuất dữ liệu ra file 'results2.csv' như yêu cầu.

* Hàm Plotting(df)

- Nạp dữ liệu vào là DataFrame (Kết quả của fuction read_data())
- Tiến hành thực hiện các bước vẽ đồ thị tổng hợp. Vẽ theo đúng 3 chỉ số tấn công, 3 chỉ số phòng thủ. Đưa kết quả ra.

- Lần lượt vẽ đồ thị theo từng chỉ số, gói dữ liệu về theo từng team để tạo đồ thị (mỗi team 1 đồ thị). Đưa dữ liệu ra.

* Hàm Best_Team_Summary(df, stats_cols)

- Nạp dữ liệu vào là DataFrame và các cột thống kê (Kết quả của fuction read_data()).
- Xử lý dữ liệu, tìm ra team có giá trị lớn nhất theo từng chỉ số.
- Đếm số lượng chỉ số dẫn đầu của mỗi team để tìm ra team có phong độ tốt nhất, từ đó đưa ra dự đoán.

2.3.2 Chi tiết các thao tác

* Thao tác read_data() (Đọc dữ liệu vào):

```
1  def read_data():
2      # Load the data
3      df = pd.read_csv(results.csv)
4
5      # Columns that are not statistics
6      non_numeric_cols = ['Name', 'Nation', 'Team', 'Position', 'Age']
7      stats_cols = [col for col in df.columns if col not in non_numeric_cols]
8
9      # Clean numeric columns (remove commas and convert to numbers)
10     for col in stats_cols:
11         df[col] = df[col].astype(str).str.replace(',', '', regex=False)
12         df[col] = pd.to_numeric(df[col], errors='coerce')
13
14     return df, stats_cols, non_numeric_cols
```

Ở thao tác này, trước hết phải lấy dữ liệu từ file: results.csv, tạo một DataFrame mới chứa dữ liệu này [dòng 3]; Xử lý, tách những dữ liệu cần thống kê (về list) và không cần thống kê [dòng 6, 7]. Sau khi đã cơ bản lấy được dữ liệu tiếp tục đi xử lý một vài vùng dữ liệu như số có dấu ',', chuyển thành chữ số [dòng 10 - 12]. Trả về kết quả là những DataFrame, list vừa xử lý.

* Thao tác Find_Top_3(df, stats_cols) (Tìm top 3 cầu thủ đứng đầu và cuối ở mỗi chỉ số)

```
1  def Find_Top_3(df, stats_cols):
2      top3_results = []
3
4      for col in stats_cols:
5          if df[col].dropna().empty:
6              continue  # Skip empty columns
7
8          # Top 3 highest
9          top_high = df[['Name', 'Team', col]].sort_values(by=col, ascending=False).head(3)
10
11         # Top 3 lowest
12         top_low = df[['Name', 'Team', col]].sort_values(by=col, ascending=True).head(3)
13
14         section = f"=== {col} ===\n"
15         section += "Top 3 Highest:\n"
16         for idx, row in top_high.iterrows():
17             section += f"  {row['Name']} ({row['Team']}): {row[col]}\n"
18
19         section += "Top 3 Lowest:\n"
20         for idx, row in top_low.iterrows():
21             section += f"  {row['Name']} ({row['Team']}): {row[col]}\n"
22
23         section += "\n"
24         top3_results.append(section)
25
26     # Save to top_3.txt
```

```
27      with open('top_3.txt', 'w', encoding='utf-8') as f:
28          f.write("\n".join(top3_results))
29
30      print("top_3.txt saved!")
```

Đầu tiên, khởi tạo một list dùng để chứa các dữ liệu đầu ra [dòng 2]. Bắt đầu chạy từng cột trong bảng những giá trị thống kê. Bỏ qua những cột trống (Tránh nhiễu dữ liệu nếu có) [dòng 5, 6]. Tạo một DataFrame mới lưu top 3 của chỉ số (Cao nhất và thấp nhất) [dòng 9, 12].
Dòng 14 đến 23 là các thao tác tạo dữ liệu để đưa vào ra (viết các mô tả để dễ dọc file đầu ra). Ở thao tác này lấy tên cầu thủ, tên team, và giá trị ở cột đang xét.
Sau khi viết xong đưa vào list đã tạo ở đầu. Tiếp tục lặp tới hết cột của bảng.
Sau khi lấy hết kết quả tiến hành ghi kết quả vào file top_3.txt như yêu cầu [dòng 27 - 29].

* Thao tác Calculate_For_Each_Statistic(df, stats_cols) (Tạo bảng thống kê các trung bình, trung vị, độ lệch của các team)

```
1  def save_df_to_file(name, res):
2      results_df.to_csv(name, index=False, encoding='utf-8-sig')
3
4  def Calculate_For_Each_Statistic(df, stats_cols):
5      # Group by Team + one overall ("all")
6      grouped = df.groupby('Team')
7      summary_rows = []
8
9      # First row: "all" players
10     summary_all = {'Team': 'all'}
11     for col in stats_cols:
12         summary_all[f'Median of {col}'] = df[col].median()
13         summary_all[f'Mean of {col}'] = df[col].mean()
14         summary_all[f'Std of {col}'] = df[col].std()
15     summary_rows.append(summary_all)
16
17     # Each team's stats
18     for team, team_df in grouped:
19         summary_team = {'Team': team}
20         for col in stats_cols:
21             summary_team[f'Median of {col}'] = team_df[col].median()
22             summary_team[f'Mean of {col}'] = team_df[col].mean()
23             summary_team[f'Std of {col}'] = team_df[col].std()
24         summary_rows.append(summary_team)
25
26     # Save to results2.csv
27     save_df_to_file('results2.csv', summary_rows)
28     print("results2.csv saved!")
```

Khởi tạo các thành phần cần thiết để xử lý dữ liệu: Tạo DataFrame gộp lại bởi các team [dòng 8]; Tạo một list lưu trữ các giá trị theo từng dòng [dòng 9].
Xử lý các giá trị ở dòng đầu (all) lặp trên từng cột của bảng thống kê tự đó đưa ra 3 giá trị như yêu cầu [Dòng 12 - 17].
Xử lý các dữ liệu của từng dòng team. Thực hiện vòng lặp để tìm từng theo từng team. Tiếp tục tạo vòng lặp theo từng trường dữ liệu đưa ra 3 chỉ số theo yêu cầu của từng trường và đưa vào list lưu kết quả. Lặp cho đến khi hết team [Dòng 20 - 26].
Đưa ra dữ liệu vào file results2.csv thông qua hàm save_df_to_file(name, res).

* Thao tác Plotting(df) (Tạo biểu đồ)

```
1  def Plotting(df):
2      # --- Setting up ---
3      attack_indexes = [
4          'Performance: goals',
5          'Performance: assists',
6          'Expected: expected goals (xG)'
7      ]
8
```

```
    defense_indexes = [
        'Tackles: Tkl',
        'Challenges: Att',
        'Blocks: Blocks'
    ]

    team_column_name = 'Team'
    max_teams_per_row_facet = 4 # How many team plots per row in FacetGrid
    all_indexes = attack_indexes + defense_indexes
    valid_indexes = []

    for col in all_indexes:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')
            if not df[col].isnull().all() and pd.api.types.is_numeric_dtype(df[col]):
                valid_indexes.append(col)

    # --- Plotting ---
    # 1. Histograms for the Entire League
    Histograms_Entire_League(df, valid_indexes)

    # 2. Histograms per Team (using FacetGrid)
    Histograms_per_Team(df, team_column_name, valid_indexes, max_teams_per_row_facet)

    print("\n--- Plotting Complete ---")
```

Ở function này bao gồm 2 thao tác chính: Chuẩn bị (được code toàn bộ trong function); Vẽ biểu đồ (ở trong 2 function phụ được gọi). Đi sâu và chi tiết:

Setting Up:

Khởi tạo các giá trị sẽ tiến hành lấy dữ liệu để vẽ biểu đồ [Dòng 3 - 15] (Bao gồm 6 thuộc tính: 3 thuộc tính về tấn công, 3 thuộc tính về phòng thủ). Tiếp tục khởi tạo một số giá trị cần thiết nhằm phục vụ cho quá trình xử lý dữ liệu và vẽ biểu đồ [dòng 15 - 18]. Thực hiện vòng lặp nhằm đưa dữ liệu về dạng số đồng thời loại bỏ những thuộc tính không thể thống kê (Đảm bảo không có lỗi khi chạy).

Plotting: Thao tác được thực hiện thông qua 2 chương trình con:

* Histograms_Entire_League(df, valid_indexes) Đồ thị tổng hợp:

```
def Histograms_Entire_League(df, valid_indexes):
    print("\n--- Plotting Overall League Distributions ---")
    num_valid_indexes = len(valid_indexes)
    # Calculate grid size for overall plots (e.g., 2 columns)
    ncols_overall = 2
    nrows_overall = (num_valid_indexes + ncols_overall - 1) // ncols_overall

    plt.figure(figsize=(12, 5 * nrows_overall))
    plt.suptitle('Overall League Distribution of Player Indexes', fontsize=16, y=1.02) #
        Add space with y

    for i, index_col in enumerate(valid_indexes):
        plt.subplot(nrows_overall, ncols_overall, i + 1)
        # Filter out NaN values for plotting if you didn't fill them earlier
        data_to_plot = df[index_col].dropna()
        if not data_to_plot.empty:
            sns.histplot(data_to_plot, kde=True, bins=20)
            plt.title(f'Distribution of {index_col}')
            plt.xlabel(index_col)
            plt.ylabel('Frequency')
        else:
            plt.title(f'{index_col}\n(No valid data to plot)')

    plt.tight_layout(rect=[0, 0, 1, 0.98]) # Adjust layout to prevent overlap with
        suptitle
    plt.savefig(os.path.join('P2_RES', 'Overall_League_Distribution_of_Player_Indexes.png
        '))
    print("\n--- Done Plotting Overall League Distributions ---")
```

Trước tiên function lấy vào cũng như khởi tạo một số giá trị để phục vụ việc bố trí. (đưa vào số đồ thị [dòng 3]; khởi tạo các thông số để bố trí hình dọc theo 2 biểu đồ 1 dòng [dòng 5, 6]). Dòng 8 tiến hành tạo khích thước cho biểu đồ mặc định là 12 x 5*(số lượng

dòng biểu đồ). Tạo tên cho ảnh ở dòng 9.
Tiến hành lặp để vẽ biểu đồ theo từng loại chỉ số được đưa vào. Dòng 12 nhằm đưa biểu đồ về đúng vị trí đã bố trí trước đó. Dòng 14 nhằm lọc bỏ những giá trị không hợp lệ, ví dụ: `'N/a'`. Nếu có giá trị thì tiến hành vẽ biểu đồ, nếu không thì tạo ô trống với định dạng tên như ở dòng 21.

Trong khi tạo biểu đồ, ta dùng lệnh: sns.histplot(data_to_plot, kde=True, bins=20) để vẽ biểu đồ histogram (biểu đồ tần suất) của dữ liệu trong `data_to_plot`.

- `data_to_plot`: là dữ liệu bạn muốn vẽ histogram, có thể là một danh sách, mảng NumPy, Series Pandas, v.v.
- `bins=20`: chia dữ liệu thành 20 khoảng (bins) để vẽ histogram. Mỗi cột trong biểu đồ đại diện cho số lượng phần tử nằm trong một khoảng nhất định.
- `kde=True`: vẽ thêm đường mật độ kernel (KDE) – một đường cong trơn ước lượng phân phối xác suất của dữ liệu, giúp bạn nhìn rõ dạng phân phối (ví dụ: chuẩn, lệch trái, lệch phải, v.v.).

Sau khi lặp xong hết các chỉ số, ta cố định layout và lưu ảnh [dòng 23, 24].
* Histograms_per_Team(df, team_column_name, valid_indexes, max_teams_per_row_facet))
Đồ thị theo từng chỉ số của mỗi team:

```
1  def Histograms_per_Team(df, team_column_name, valid_indexes, max_teams_per_row_facet):
2      print("\n--- Plotting Per-Team Distributions ---")
3
4      # Check number of unique teams to avoid overly large grids
5      unique_teams = df[team_column_name].nunique()
6      print(f"Found {unique_teams} unique teams.")
7      if unique_teams > 50: # Add a threshold to prevent overwhelming plots
8          print("Warning: High number of teams detected. FacetGrid might be very large.")
9          # Optional: Add logic here to maybe plot only a subset of teams or ask user
10
11     for index_col in valid_indexes:
12         print(f"Generating FacetGrid for: {index_col}")
13
14         # Filter out NaNs for this specific index and the team column before creating the
               grid
15         facet_data = df[[index_col, team_column_name]].dropna()
16
17         if facet_data.empty or facet_data[index_col].isnull().all():
18             print(f"  Skipping {index_col} - No valid data after dropping NaNs.")
19             continue
20
21         # Create the FacetGrid
22         g = sns.FacetGrid(
23             facet_data,
24             col=team_column_name,
25             col_wrap=min(max_teams_per_row_facet, unique_teams), # Don't wrap more than
                   teams exist
26             sharex=True, # Keep x-axis consistent for comparison
27             sharey=False, # Allow y-axis (frequency) to vary per team
28             height=3,     # Adjust height of each subplot
29             aspect=1.2    # Adjust aspect ratio of each subplot
30         )
31
32         # Map the histogram plot onto the grid
33         g.map(sns.histplot, index_col, kde=True, bins=15) # Use fewer bins for smaller
               plots
34
35         # Add titles and adjust layout
36         g.set_titles("Team: {col_name}")
37         g.fig.suptitle(f'Distribution of {index_col.split} by Team', fontsize=14, y=1.03)
               # Add overall title slightly above
38         g.fig.tight_layout(rect=[0, 0, 1, 0.97]) # Adjust layout
39         name = (index_col.replace(' ', '_')).replace(':', '')
40         plt.savefig(os.path.join('P2_RES','Distribution_of_'+name+'_by_Team.png'))
41     print("\n--- Done Plotting Per-Team Distributions ---")
```

- Ban đầu, hàm in thông báo bắt đầu quá trình vẽ biểu đồ và xác định số lượng đội bóng khác nhau trong dữ liệu đầu vào. Nếu số lượng đội quá lớn, chương trình sẽ đưa ra cảnh báo về nguy cơ quá tải biểu đồ (dòng 5–8).
- Sau đó, hàm lặp qua từng chỉ số trong danh sách `valid_indexes`. Với mỗi chỉ số, dữ liệu được lọc để chỉ giữ lại những giá trị hợp lệ tương ứng với chỉ số và tên đội (dòng 11–12). Nếu toàn bộ dữ liệu của chỉ số đó là thiếu (`NaN`), chỉ số sẽ bị bỏ qua (dòng 17–18).
- Đối với các chỉ số có dữ liệu hợp lệ, hàm tạo một lưới biểu đồ (`FacetGrid`) trong đó mỗi ô biểu diễn phân phối của chỉ số theo từng đội bóng. Lưới được điều chỉnh với chiều cao, tỉ lệ, và số cột phù hợp, đồng thời giữ cố định trục hoành giữa các biểu đồ để thuận tiện cho việc so sánh (dòng 22–30).
- Biểu đồ histogram với 15 khoảng (`bins`) cùng với đường mật độ KDE được ánh xạ lên từng ô trong lưới (dòng 33). Mỗi ô có tiêu đề phụ hiển thị tên đội, đồng thời tiêu đề chung cho toàn bộ lưới được thêm phía trên cùng. Hàm sử dụng `tight_layout()` để căn chỉnh bố cục tránh chồng lấn (dòng 36–38).
- Cuối cùng, biểu đồ được lưu vào thư mục `P2_RES` với tên tệp động dựa trên chỉ số đang xử lý (dòng 40), và thông báo kết thúc quá trình vẽ biểu đồ được in ra (dòng 41).

* Thao tác Best_Team_Summary(df, stats_cols): Tìm ra team tốt nhất ở mỗi chỉ số và dự đoán team có thành tích tốt nhất

```
def Best_Team_Summary(df, stats_cols):
    # Group by Team
    grouped_team = df.groupby('Team')

    # Store the best team per stat
    best_team_per_stat = {}

    for col in stats_cols:
        if df[col].dropna().empty:
            continue
        # Calculate mean per team
        mean_per_team = grouped_team[col].mean()
        best_team = mean_per_team.idxmax()  # Team with highest mean
        best_score = mean_per_team.max()
        best_team_per_stat[col] = (best_team, best_score)

    # Count how many times each team was best
    from collections import Counter
    team_counter = Counter([team for team, score in best_team_per_stat.values()])

    # Find the team that was best most often
    best_overall_team, count = team_counter.most_common(1)[0]

    # Save results
    file_path = os.path.join('P2_RES', 'best_team_summary.txt')
    data = []

    for stat, (team, score) in best_team_per_stat.items():
        data.append({
            'Statistic': stat,
            'Best Team': team,
            'Average Score': round(score, 2)
        })

    df = pd.DataFrame(data)

    overall_row = {
        'Statistic': 'Best Overall Team',
        'Best Team': best_overall_team,
```

```
40          'Average Score': f'Top in {count} statistics'
41      }
42
43      df = pd.concat([df, pd.DataFrame([overall_row])], ignore_index=True)
44      df.to_csv(file_path, index=False, sep='\t')
45      print(f"Best team identified: {best_overall_team} (Top in {count} stats). See '
            best_team_summary.txt'.")
```

- Trước tiên, dữ liệu được nhóm theo cột `'Team'` nhằm phục vụ cho việc tính toán giá trị trung bình theo từng đội (dòng 3). Một từ điển `best_team_per_stat` được khởi tạo để lưu đội có chỉ số cao nhất cho từng chỉ số (dòng 6).
- Hàm tiếp tục lặp qua từng chỉ số trong danh sách `stats_cols`. Nếu một cột chỉ chứa toàn bộ giá trị thiếu (`NaN`), chỉ số đó sẽ bị bỏ qua (dòng 9–10). Với mỗi chỉ số hợp lệ, giá trị trung bình theo từng đội được tính, và đội có giá trị trung bình cao nhất được xác định bằng phương thức `idxmax()` (dòng 12–14). Cặp thông tin gồm tên đội và giá trị trung bình được lưu vào từ điển kết quả (dòng 15).
- Tiếp theo, hàm sử dụng `collections.Counter` để đếm số lần mỗi đội được chọn là đội tốt nhất trong các chỉ số (dòng 18–18). Đội được xem là dẫn đầu tổng thể là đội xuất hiện nhiều nhất trong các lựa chọn trên (dòng 22).
- Sau đó, kết quả được chuẩn bị để ghi vào tệp văn bản. Một danh sách `data` được tạo, trong đó mỗi phần tử là một dòng gồm tên chỉ số, tên đội dẫn đầu và điểm trung bình (dòng 25–35). Một dòng tổng kết (`overall_row`) được thêm vào, ghi nhận đội dẫn đầu tổng thể cùng số lượng chỉ số mà đội đó dẫn đầu (dòng 37–41).
- Cuối cùng, toàn bộ thông tin được đóng gói thành một `DataFrame` và được xuất ra tệp `best_team_summary.txt` theo định dạng TSV (giá trị phân cách bằng dấu tab) trong thư mục `P2_RES` (dòng 43–44). Một thông báo kết quả được in ra màn hình để xác nhận hoàn tất quá trình xử lý (dòng 45).

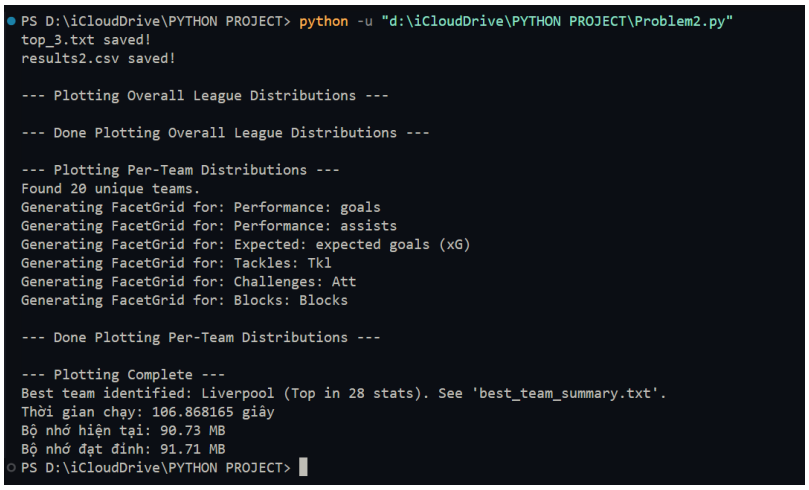
2.4 Kết quả và Đánh giá

2.4.1 Kết quả

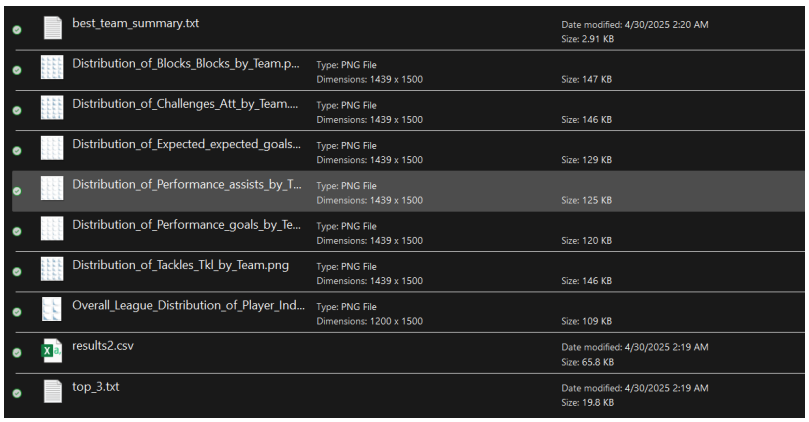
Kết quả chung

Đoạn terminal cho thấy quá trình thực thi thành công của script Python `Problem2.py`. Trong quá trình chạy, chương trình đã lưu hai tệp kết quả là `top_3.txt` và `results2.csv`, sau đó thực hiện vẽ biểu đồ phân phối dữ liệu cho toàn giải đấu và cho từng đội bóng. Có tổng cộng 20 đội được nhận diện, và biểu đồ được tạo cho nhiều chỉ số như `goals`, `assists`, `expected goals (xG)`, `tackles (Tkl)`, `challenges (Att)` và `blocks`. Kết quả phân tích cho thấy đội `Liverpool` là đội xuất sắc nhất, dẫn đầu ở 28 chỉ số. Thông tin chi tiết được lưu vào tệp `best_team_summary.txt`. Quá trình xử lý hoàn tất sau khoảng 106.87 giây với mức sử dụng bộ nhớ tối đa là 91.71 MB.

Đầu ra chương trình gồm có 10 tệp, trong đó có 2 tệp đuôi txt, 1 tệp đuôi csv, 7 tệp đuôi png. Ở tệp top_3.txt chứa dữ liệu đầu ra của function tìm top 3 người cao nhất và thấp nhất của mỗi chỉ số. Tệp results2.csv là đầu ra của function thống kê trung bình, trung vị và độ lệch của mỗi team theo yêu cầu đầu ra. Các file đuôi png là hình gồm 1 hình tổng hợp 6 thuộc tính cảu tất cả các cầu thủ chứa 6 biểu đồ, 6 hình còn lại mỗi hình



Hình 2.1: Terminal sau khi chạy chương trình Problem2.py



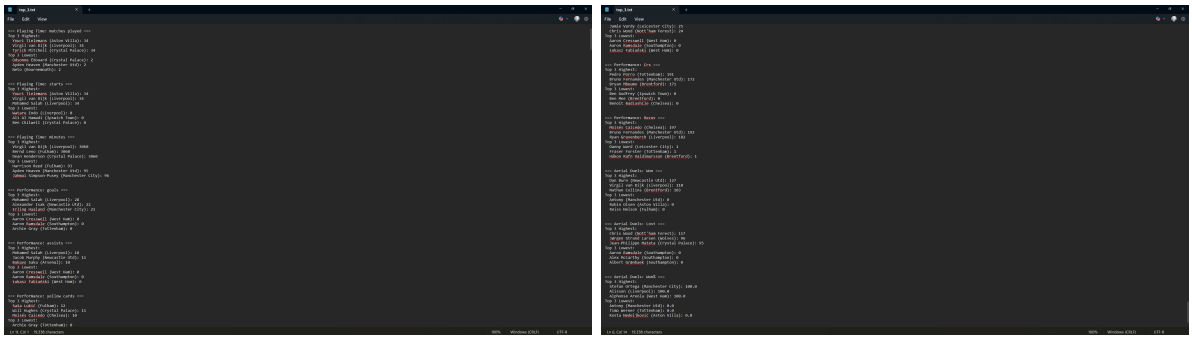
Hình 2.2: Các tệp đầu ra sau khi chạy chương trình Problem2.py

chứa 20 biểu đồ là thống kê theo 6 thuộc tính của mỗi team. Đây cũng là kết quả đầu ra chủa chương trình vẽ đồ thị. File txt cuối cùng là kết quả đầu ra của chương trình tìm ra team cao nhất ở mỗi thuộc tính và dự đoán team có thành tích tốt nhất của mùa giải.

Chi tiết kết quả đầu ra

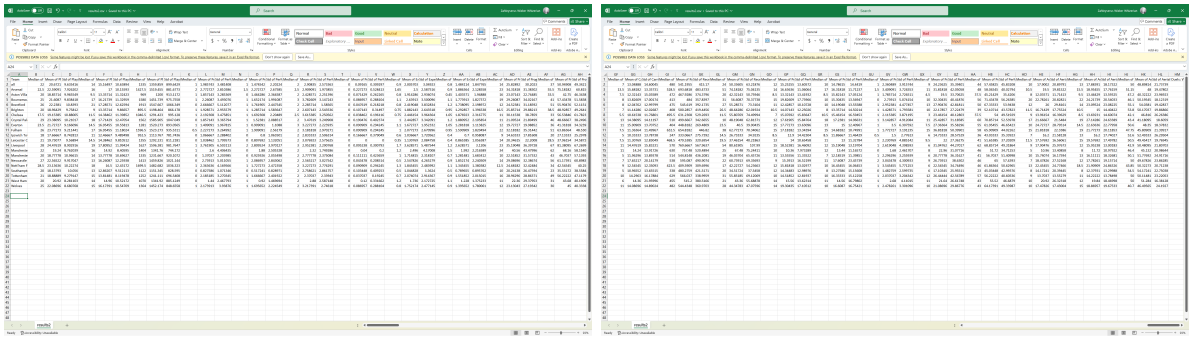
top_3.txt Tệp này cung cấp một phân tích định lượng chi tiết về hiệu suất của cầu thủ trong một giải đấu bóng đá, được sắp xếp theo các danh mục thống kê. Mỗi danh mục so sánh "Top 3 Highest"và "Top 3 Lowest"để làm nổi bật phạm vi hiệu suất. Các danh mục bao gồm thời gian thi đấu, hiệu suất (ví dụ: bàn thắng, kiến tạo), chỉ số kỳ vọng (ví dụ: xG, xAG), chỉ số tiến triển, số liệu thống kê trên 90 phút, số liệu thống kê tiêu chuẩn, số liệu thống kê về chuyền bóng, các đường chuyền quan trọng, các hành động tạo ra cú sút, tắc bóng và tranh chấp, đánh chặn và phá bóng, chạm bóng, rê bóng, chuyền bóng, nhận bóng và phạm lỗi. Nhìn chung, tệp cung cấp một cái nhìn có cấu trúc về điểm mạnh và điểm yếu của cầu thủ.

results2.csv Tệp results2.csv chứa dữ liệu thống kê tổng hợp về các đội bóng trong một giải đấu, với tổng cộng 21 đội (hoặc nhóm) và 220 cột đặc trưng liên quan đến hiệu suất thi đấu. Mỗi hàng đại diện cho một đội, bao gồm tên đội và các chỉ số mô tả trung vị, trung bình, độ lệch chuẩn của nhiều khía cạnh trong thi đấu.
Các thông số được ghi nhận bao gồm:



Hình 2.3: File top_3.txt

- Thời gian thi đấu: số trận ra sân, số trận đá chính, tổng phút thi đấu;
- Chỉ số thi đấu kỹ thuật: như số pha không chiến thắng/thua, tỷ lệ thắng không chiến;
- Dạng thống kê: mỗi chỉ số thường có 3 biến đi kèm – giá trị trung vị (Median), trung bình (Mean) và độ lệch chuẩn (Std), thể hiện sự phân tán trong tập dữ liệu.

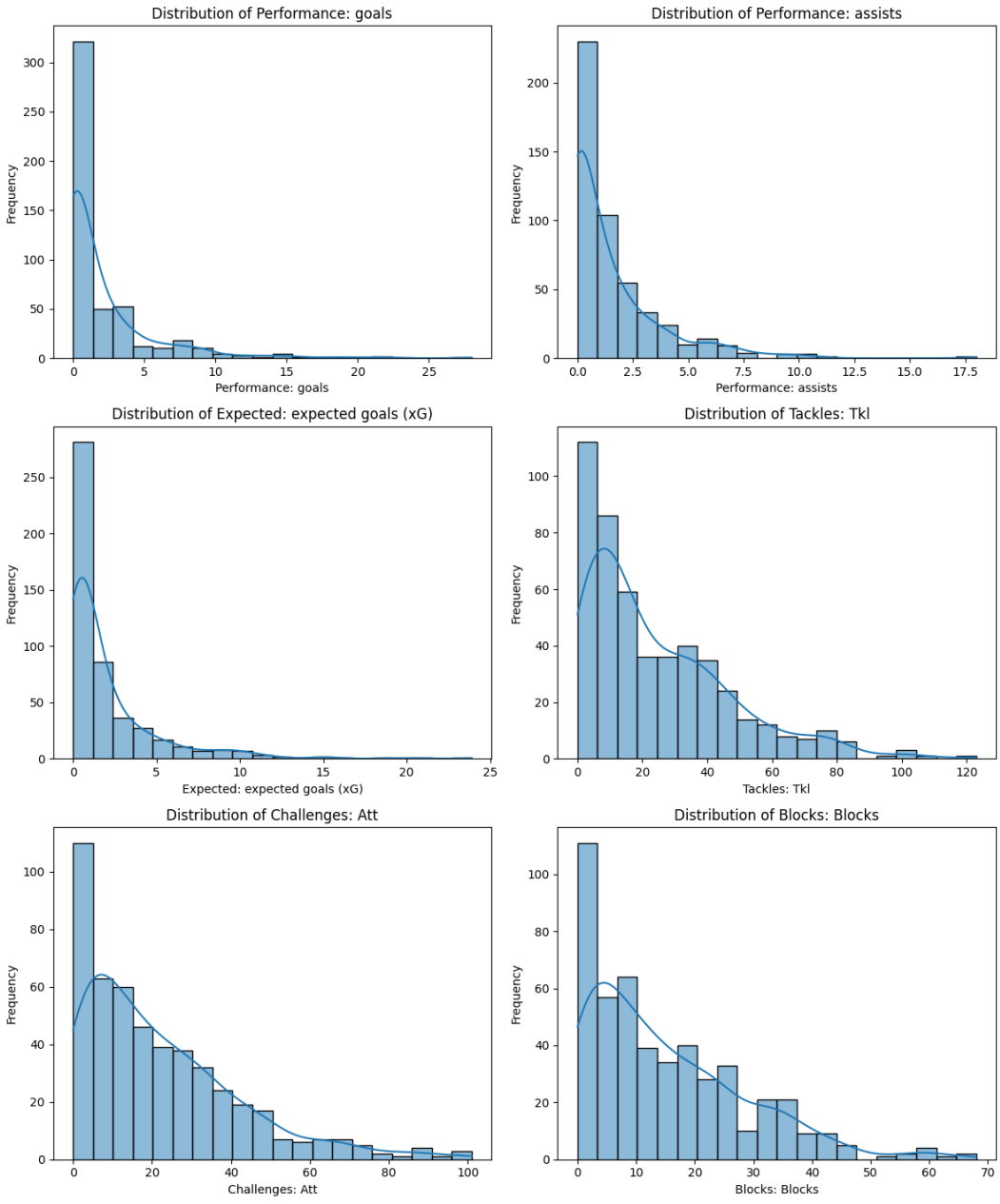


Hình 2.4: File results2.csv

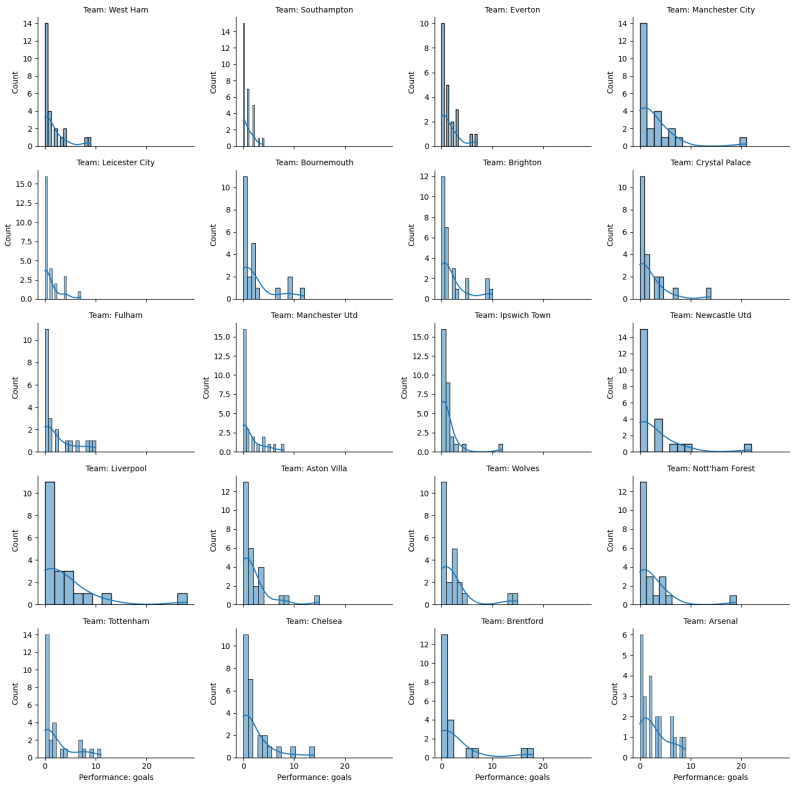
Overall_League_Distribution_of_Player_Indexes.png Hình ảnh gồm 6 biểu đồ histogram được bố trí theo dạng lưới 2 cột × 3 hàng, mỗi biểu đồ có:

- Tiêu đề rõ ràng ở phía trên, mô tả tên chỉ số được thể hiện (ví dụ: Distribution of Performance: goals).
- Trục hoành (X-axis) hiển thị tên chỉ số cụ thể (như Performance: goals, Tackles: Tkl).
- Trục tung (Y-axis) được gán nhãn là Frequency, biểu thị tần suất xuất hiện.
- Các cột histogram màu xanh nhạt minh họa phân bố dữ liệu.

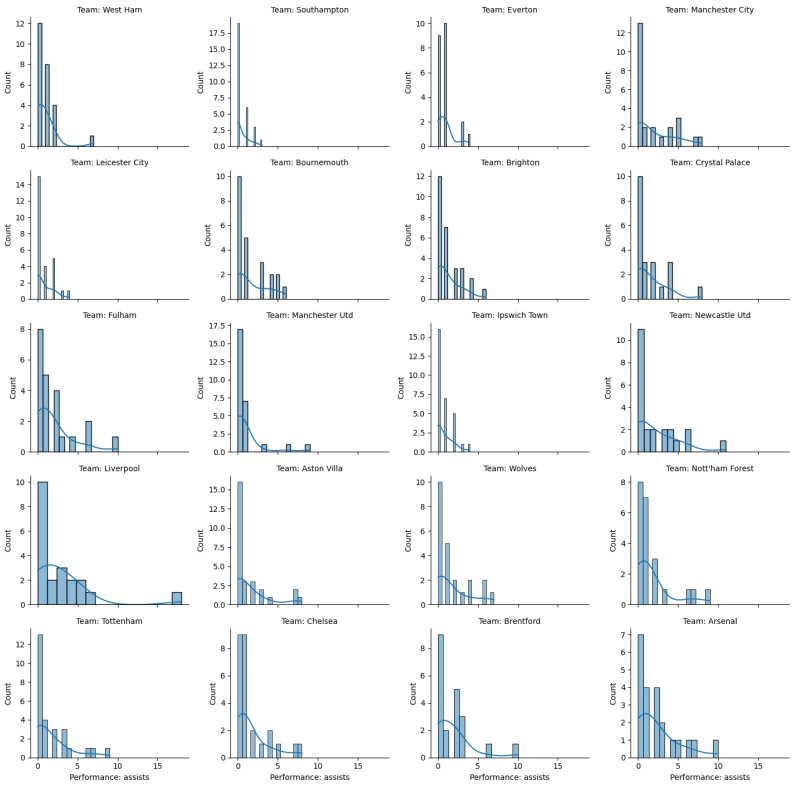
Một đường cong KDE màu xanh đậm được vẽ mượt theo phân bố dữ liệu để trực quan hóa mật độ xác suất. Toàn bộ hình ảnh có bố cục gọn gàng, đồng nhất về kiểu trình bày và định dạng trục, giúp dễ dàng so sánh giữa các biểu đồ.



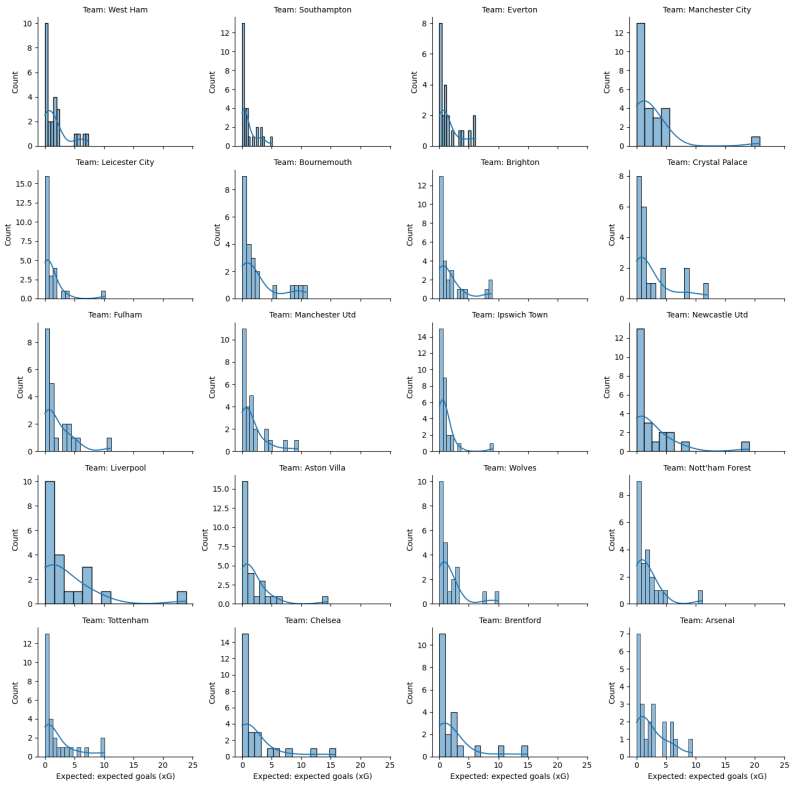
Hình 2.5: File Overall_League_Distribution_of_Player_Indexes.png



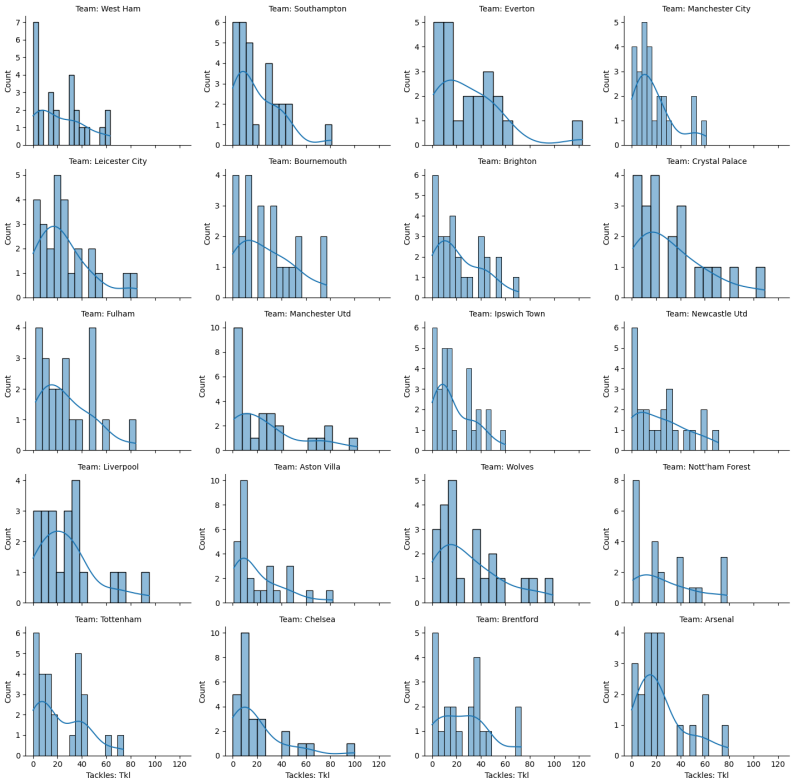
Hình 2.6: File Distribution_of_Performance_goals_by_Team.png



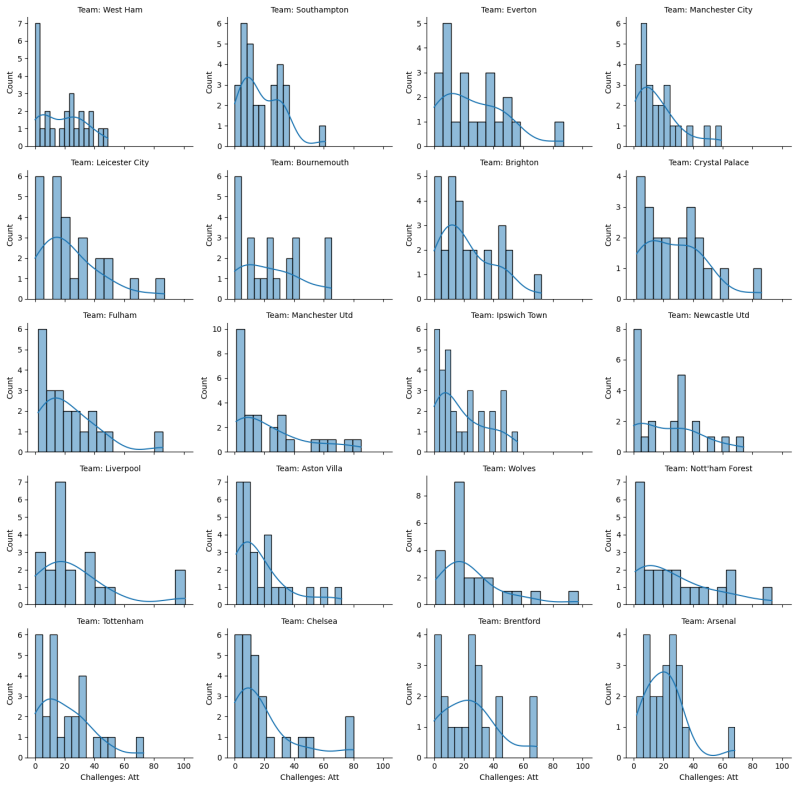
Hình 2.7: File Distribution_of_Performance_assists_by_Team.png



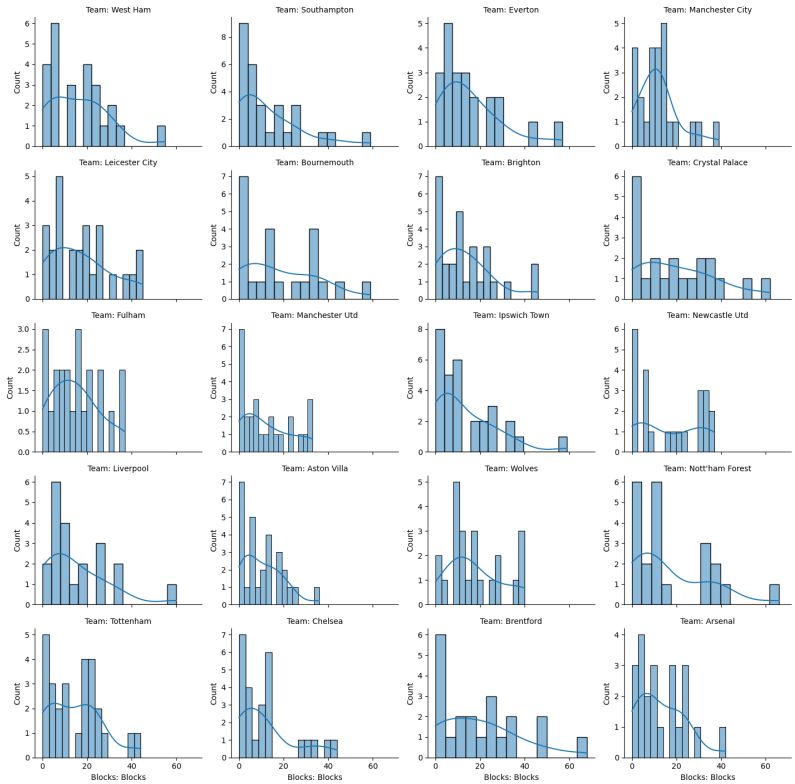
Hình 2.8: File Distribution_of_Expected_expected_goals_(xG)_by_Team.png



Hình 2.9: File Distribution_of_Tackles_Tkl_by_Team.png



Hình 2.10: File Distribution_of_Challenges_Att_by_Team.png



Hình 2.11: File Distribution_of_Blocks_Blocks_by_Team.png

Distribution_of_Performance_goals_by_Team.png: Hình ảnh thể hiện dưới dạng ma trận lưới biểu đồ histogram gồm 24 biểu đồ con (4 hàng × 6 cột), mỗi biểu đồ đại diện cho một đội bóng khác nhau trong giải đấu.
Về hình thức:

- Tiêu đề mỗi biểu đồ nằm phía trên cùng, ghi rõ tên đội (ví dụ: Team: Manchester City, Team: Arsenal).
- Trục hoành (X-axis) được gắn nhãn nhất quán là Performance: goals, thể hiện số bàn thắng.
- Trục tung (Y-axis) được đánh dấu là Count, biểu thị số lượng cầu thủ đạt mức bàn thắng tương ứng.
- Cột histogram màu xanh lam nhạt, kèm theo đường cong KDE màu xanh đậm nhằm biểu diễn mật độ xác suất phân bố bàn thắng của các cầu thủ trong từng đội.

Toàn bộ bố cục được sắp xếp đều đặn, đồng nhất về tỷ lệ và kiểu dáng, giúp dễ dàng so sánh sự phân bố bàn thắng giữa các đội bóng. Đây là kiểu trực quan hóa thường dùng trong báo cáo thống kê thể thao hoặc phân tích dữ liệu định lượng đa nhóm.

Distribution_of_Performance_assists_by_Team.png: Hình ảnh này là một bố cục trực quan dạng lưới gồm 24 biểu đồ histogram nhỏ, mỗi biểu đồ đại diện cho một đội bóng khác nhau trong giải đấu. Về mặt hình thức, hình ảnh có các đặc điểm sau:

- Cấu trúc lưới 6 cột × 4 hàng, giúp tổ chức biểu đồ một cách đều đặn và dễ theo dõi.
- Mỗi biểu đồ con có tiêu đề ở phía trên ghi rõ tên đội bóng, ví dụ: Team: Arsenal, Team: Manchester Utd, v.v.
- Trục hoành ở tất cả các biểu đồ được gán nhãn là Performance: goals, thể hiện số bàn thắng mà cầu thủ ghi được.
- Trục tung có nhãn là Count, biểu diễn số lượng cầu thủ tương ứng với từng mức bàn thắng.
- Trong mỗi biểu đồ, các cột histogram màu xanh lam nhạt biểu thị tần suất, đi kèm với một đường cong KDE màu xanh đậm, thể hiện mật độ phân bố xác suất.
- Kiểu dáng, tỷ lệ trục và bố cục biểu đồ được giữ nhất quán giữa các đội, tạo nên một tổng thể trực quan rõ ràng, dễ dàng để so sánh phân bố bàn thắng giữa các đội bóng.

Tổng thể, hình ảnh được trình bày theo phong cách khoa học và trực quan hóa dữ liệu chuẩn mực, phù hợp với các báo cáo học thuật hoặc phân tích thống kê định lượng trong lĩnh vực thể thao.

Distribution_of_Expected_expected_goals_(xG)_by_Team.png: Hình ảnh này là một bố cục trực quan dạng lưới gồm 24 biểu đồ histogram nhỏ, mỗi biểu đồ đại diện cho một đội bóng khác nhau trong giải đấu. Về mặt hình thức, hình ảnh có các đặc điểm sau:

- Cấu trúc lưới 6 cột × 4 hàng, giúp tổ chức biểu đồ một cách đều đặn và dễ theo dõi.
- Mỗi biểu đồ con có tiêu đề ở phía trên ghi rõ tên đội bóng, ví dụ: Team: Arsenal, Team: Manchester Utd, v.v.
- Trục hoành ở tất cả các biểu đồ được gán nhãn là Performance: expected_goals (xG), thể hiện số bàn thắng kỳ vọng mà cầu thủ tạo ra.
- Trục tung có nhãn là Count, biểu diễn số lượng cầu thủ tương ứng với từng mức xG.
- Trong mỗi biểu đồ, các cột histogram màu xanh lam nhạt biểu thị tần suất, đi kèm với một đường cong KDE màu xanh đậm, thể hiện mật độ phân bố xác suất.
- Kiểu dáng, tỷ lệ trục và bố cục biểu đồ được giữ nhất quán giữa các đội, tạo nên một tổng thể trực quan rõ ràng, dễ dàng để so sánh phân bố xG giữa các đội bóng.

Tổng thể, hình ảnh được trình bày theo phong cách khoa học và trực quan hóa dữ liệu chuẩn mực, phù hợp với các báo cáo học thuật hoặc phân tích thống kê định lượng trong lĩnh vực thể thao.

Distribution_of_Tackles_Tkl_by_Team.png: Hình ảnh là một bố cục trực quan gồm 24 biểu đồ nhỏ sắp xếp dạng lưới 6x4.

- Tiêu đề từng biểu đồ con xác định rõ tên đội.
- Trục hoành là Performance: tackles_tkl, phản ánh số pha tắc bóng của cầu thủ.
- Trục tung là Count, cho biết số cầu thủ đạt mức tắc bóng tương ứng.
- Histogram xanh lam nhạt và KDE xanh đậm tiếp tục là hình thức thể hiện chính.
- Sự nhất quán về tỷ lệ, màu sắc, bố cục và định dạng tiếp tục được duy trì.

Biểu đồ mang tính trực quan hóa mạnh cho các thống kê phòng ngự chủ động (tắc bóng) của các đội bóng.

Distribution_of_Challenges_Att_by_Team.png: Hình ảnh trực quan này cũng gồm 24 biểu đồ nhỏ sắp xếp theo lưới 6x4, mỗi biểu đồ đại diện cho một đội bóng.

- Tiêu đề mỗi biểu đồ con đều theo định dạng Team: [Tên đội].
- Trục hoành ghi là Performance: challenges_att, biểu thị số lần tham gia tranh chấp bóng.
- Trục tung là Count, phản ánh số cầu thủ ứng với mỗi mức độ thử thách.
- Cột histogram xanh lam nhạt kết hợp đường KDE xanh đậm vẫn là phương pháp thể hiện chính.
- Tỷ lệ và khung hình giữ sự nhất quán tuyệt đối giữa các đội, giúp so sánh dễ dàng.

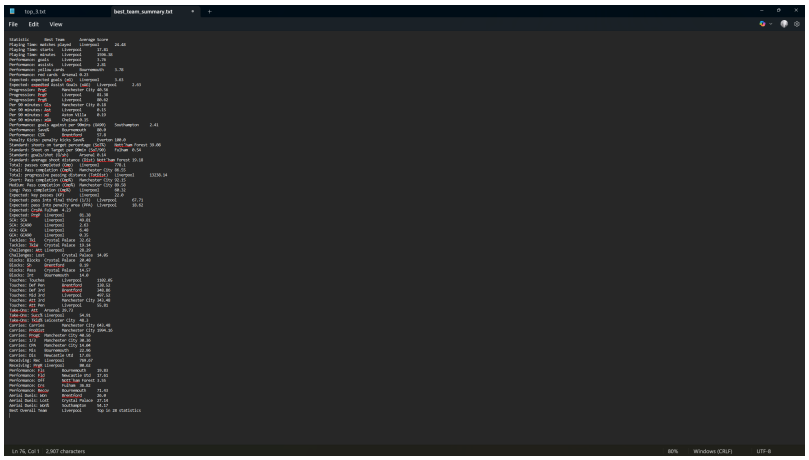
Hình ảnh mang tính trực quan hóa tốt cho các thống kê liên quan đến sức mạnh tranh chấp của từng đội.

Distribution_of_Blocks_Blocks_by_Team.png: Hình ảnh này là một bố cục trực quan dạng lưới gồm 24 biểu đồ histogram nhỏ, mỗi biểu đồ đại diện cho một đội bóng khác nhau. Về mặt hình thức:

- Cấu trúc dạng lưới 6 cột × 4 hàng được duy trì.
- Tiêu đề trên mỗi biểu đồ con thể hiện tên đội bóng, ví dụ: Team: Chelsea, Team: Leeds, v.v.
- Trục hoành được gán nhãn là Performance: blocks, thể hiện số lần cầu thủ thực hiện các pha cản phá.
- Trục tung là Count, cho biết số lượng cầu thủ tương ứng với từng mức blocks.
- Histogram màu xanh lam nhạt biểu thị tần suất, kết hợp đường KDE xanh đậm để thể hiện xu hướng phân bố.
- Bố cục, tỷ lệ và kiểu dáng đồng bộ, đảm bảo dễ dàng so sánh giữa các đội.

Tổng thể hình ảnh giữ phong cách chuyên nghiệp, có tính nhất quán cao, hỗ trợ hiệu quả cho việc phân tích phòng ngự giữa các đội.

best_team_summary.txt: Tệp best_team_summary.txt này chứa một bản tóm tắt



Hình 2.12: File best_team_summary.txt

các số liệu thống kê về các đội bóng đá, có vẻ như là từ một giải đấu cụ thể. Dữ liệu được phân loại theo nhiều khía cạnh khác nhau của trận đấu như Tuổi (Age), Thời gian thi đấu (Playing Time), Hiệu suất (Performance), Chỉ số kỳ vọng (Expected), Khả năng phát triển bóng (Progression), Chỉ số tính trên 90 phút (Per 90 minutes), Thống kê về chuyền bóng (Total, Short, Medium, Long), Thống kê về phòng ngự (Tackles, Challenges, Blocks), Chạm bóng (Touches), Qua người (Take-Ons), Kéo bóng (Carries), Nhận bóng (Receiving), và Không chiến (Aerial Duels).
Đối với mỗi chỉ số thống kê, tệp liệt kê tên đội bóng dẫn đầu hạng mục đó cùng với giá trị trung bình (Avg) của đội đó cho chỉ số tương ứng. Ví dụ, Fulham có độ tuổi trung bình

cao nhất (28.27), Liverpool dẫn đầu ở nhiều chỉ số tấn công như số trận đã đấu, số phút thi đấu, bàn thắng, kiến tạo, đường chuyền vào 1/3 cuối sân và vào vòng cấm, trong khi Manchester City nổi bật ở các chỉ số liên quan đến chuyền bóng và kéo bóng tịnh tiến. Các đội khác như Bournemouth, Arsenal, Crystal Palace, Brentford cũng dẫn đầu ở một số hạng mục cụ thể như thẻ vàng, thẻ đỏ, tắc bóng, chặn bóng.
Cuối cùng, tệp kết luận rằng Liverpool là "Đội bóng xuất sắc nhất toàn diện"(Best Overall Team) vì dẫn đầu trong 28 chỉ số thống kê. Đây cũng chính là câu trả lời cho câu hỏi: "Based on your analysis, which team do you think is performing the best in the 2024-2025 Premier League season?".

2.4.2 Đánh giá

Đoạn mã Problem2.py thể hiện nhiều ưu điểm nổi bật trong thiết kế và triển khai. Cấu trúc chương trình được tổ chức hợp lý với các hàm phân tách rõ ràng theo từng chức năng như tiền xử lý dữ liệu, phân tích thống kê mô tả, trực quan hóa và xuất kết quả, giúp nâng cao tính dễ hiểu và khả năng bảo trì. Mã sử dụng hiệu quả các thư viện như Pandas và Seaborn để xử lý dữ liệu và vẽ biểu đồ, từ đó hỗ trợ phân tích trực quan và toàn diện. Đáng chú ý, chương trình tích hợp đo lường thời gian chạy và bộ nhớ sử dụng thông qua thư viện tracemalloc, phản ánh rõ sự quan tâm đến hiệu suất thực thi. Cụ thể, thời gian thực thi được ghi nhận là 1.678 giây, với mức sử dụng bộ nhớ hiện tại là 6.66 MB và mức đỉnh là 42.91 MB, cho thấy chương trình vận hành hiệu quả và phù hợp với các hệ thống có giới hạn tài nguyên.
Tuy nhiên, đoạn mã vẫn tồn tại một số hạn chế. Việc lựa chọn đội bóng xuất sắc nhất hiện chỉ dựa trên số lượng chỉ số mà đội đó dẫn đầu, trong khi chưa xem xét đến độ lớn của sự khác biệt hoặc tầm quan trọng tương đối giữa các chỉ số, điều này có thể dẫn đến kết quả thiếu khách quan trong một số trường hợp. Ngoài ra, kết quả đầu ra chủ yếu là các tệp văn bản và biểu đồ tĩnh, chưa hỗ trợ tương tác, điều này phần nào giới hạn trải nghiệm người dùng cuối và tiềm năng mở rộng ứng dụng lên các nền tảng trực tuyến hoặc hệ thống dashboard động.

Chương 3

VẤN ĐỀ III

3.1 Yêu cầu

- Use the K-means algorithm to classify players into groups based on their statistics.
- How many groups should the players be classified into? Why? Provide your comments on the results.
- Use PCA to reduce the data dimensions to 2, then plot a 2D cluster of the data points.

3.2 Các bước tiến hành

1. **Tải và tiền xử lý dữ liệu:**
    - Đọc dữ liệu từ file `results.csv` (kết quả từ Vấn đề 1).
    - Tách các cột định danh (Tên, Quốc tịch, Đội, Vị trí, Tuổi) và các cột chứa chỉ số thống kê.
    - Chuyển đổi các cột thống kê sang dạng số, xử lý các giá trị không phải số bằng phương pháp `coerce`.
    - Loại bỏ các cột chỉ chứa giá trị NaN.
    - Xử lý các giá trị bị thiếu (NaN) bằng cách thay thế bằng trung vị (`median`) của cột tương ứng, sử dụng `SimpleImputer`.
    - Chuẩn hóa dữ liệu bằng `StandardScaler` để đưa các chỉ số về cùng thang đo – điều này quan trọng đối với K-means.
2. **Xác định số cụm tối ưu (K):**
    - Sử dụng phương pháp Elbow để tính tổng bình phương sai số trong cụm (`inertia`) cho các giá trị K từ 2 đến 15.
    - Vẽ biểu đồ Elbow (`Inertia vs. K`) để xác định điểm "khuỷu tay", nơi tốc độ giảm inertia chậm lại đáng kể.
    - Tính `Silhouette Score` cho các giá trị K từ 2 đến 15 để đo lường sự phân tách giữa các cụm.

- Vẽ biểu đồ `Silhouette Score vs. K`. Giá trị K cho Silhouette Score cao thường là lựa chọn tốt.
- Lưu các biểu đồ này thành file ảnh.
- Dựa trên cả hai biểu đồ để chọn số cụm tối ưu (ví dụ: $K = 6$).

3. **Áp dụng thuật toán K-means:**
   - Thực hiện phân cụm K-means với số cụm tối ưu đã chọn ($K =$ `optimal_k`) trên dữ liệu đã chuẩn hóa.
   - Gán nhãn cụm cho từng cầu thủ.
   - Thêm cột `Cluster` vào cả DataFrame gốc (`df`) và DataFrame đã chuẩn hóa (`df_scaled`).
   - Phân tích đặc điểm cụm bằng cách tính trung bình các chỉ số thống kê (chuẩn hóa và gốc) cho từng cụm.
4. **Giảm chiều dữ liệu bằng PCA:**
   - Áp dụng PCA để giảm dữ liệu chuẩn hóa xuống còn 2 thành phần chính.
   - Tạo DataFrame mới chứa 2 thành phần chính (PC1, PC2) và nhãn cụm tương ứng.
5. **Trực quan hóa cụm 2D:**
   - Vẽ biểu đồ scatter plot 2D từ 2 thành phần PCA, tô màu điểm dữ liệu theo cụm K-means.
   - Lưu biểu đồ thành file ảnh.
6. **Lưu nhận xét và kết quả:**
   - Tạo file văn bản `comment_P3.txt` ghi lại lý do chọn K, giải thích biểu đồ PCA và phân tích thành phần cụm.

3.3 Code thực tế và mô tả chi tiết

3.3.1 Hàm chính

```
def Problem_3():
    df_scaled, df, df_imputed = Load_and_Preprocess_Data()
    Determine_Optimal_K(df_scaled)
    optimal_k = 6
    print(f"\nBased on the Elbow method and Silhouette Score plots, choosing K = {
        optimal_k}.")
    cluster_labels = Apply_K_means(df_scaled, df, optimal_k, df_imputed)
    pca_clusters_df = Apply_PCA(df_scaled, cluster_labels)
    Plot_2D_Cluster(pca_clusters_df, optimal_k)

    #Create a file comments for the results
    with open(...) as f:
      .....
```

Code được tổ chức thành các hàm trong file `Problem3.py`:

- `Load_and_Preprocess_Data()`: Tải dữ liệu, xử lý thiếu, chuẩn hóa. Trả về `df_scaled`, `df`, `df_imputed`.

- `Determine_Optimal_K(df_scaled)`: Vẽ biểu đồ Elbow và Silhouette Score, lưu ảnh vào thư mục `P3_RES`.
- `Apply_K_means(df_scaled, df, optimal_k, df_imputed)`: Chạy K-means, gán nhãn cụm, in trung bình chỉ số.
- `Apply_PCA(df_scaled, cluster_labels)`: Áp dụng PCA và trả về DataFrame chứa PC1, PC2, Cluster.
- `Plot_2D_Cluster(pca_clusters_df, optimal_k)`: Vẽ biểu đồ 2D, lưu ảnh.
- `Problem_3()`: Hàm chính điều phối toàn bộ quy trình, giả định chọn `optimal_k = 6`, lưu kết quả và nhận xét vào thư mục `P3_RES`.

3.3.2 Chi tiết các thao tác

Hàm `Load_and_Preprocess_Data()`

```
1  def Load_and_Preprocess_Data():
2      Exclude_Cols = ['Name', 'Nation', 'Team', 'Position', 'Age']
3
4      file_path = os.path.join('P1_RES', 'results.csv')
5      df = pd.read_csv(file_path)
6      # --- Preprocessing ---
7      identifiers = df[Exclude_Cols]
8      features = df.drop(columns=Exclude_Cols).apply(pd.to_numeric, errors='coerce')
9
10     cols_all_nan = features.columns[features.isnull().all()]
11     if not cols_all_nan.empty:
12         features = features.drop(columns=cols_all_nan)
13
14     imputer = SimpleImputer(strategy='median')
15     scaler = StandardScaler()
16
17     df_imputed = pd.DataFrame(imputer.fit_transform(features), columns=features.columns)
18     df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df_imputed.columns
           )
19
20     return df_scaled, df, df_imputed
```

Trong dòng 2, các cột định danh bao gồm `'Name'`, `'Nation'`, `'Team'`, `'Position'` và `'Age'` được liệt kê trong danh sách `Exclude_Cols` nhằm loại trừ khỏi quá trình tiền xử lý dữ liệu. Ở dòng 4, đường dẫn tới tệp dữ liệu `'results.csv'` được xây dựng bằng cách kết hợp thư mục `'P1_RES'` với tên tệp. Dữ liệu sau đó được đọc vào một `DataFrame df` bằng `pd.read_csv()` (dòng 5).

Tiếp theo, dòng 7 tách các cột định danh khỏi `DataFrame` gốc để lưu trong biến `identifiers`, trong khi phần còn lại của dữ liệu – các đặc trưng số – được lưu vào `features`. Ở đây, `pd.to_numeric(..., errors='coerce')` được sử dụng để chuyển đổi các giá trị không hợp lệ thành `NaN`, đảm bảo dữ liệu đầu vào cho các bước xử lý tiếp theo là dạng số.

Dòng 10-12 kiểm tra xem có cột nào chứa toàn giá trị thiếu (`NaN`) hay không. Nếu có, các cột này được loại bỏ khỏi `features` để tránh gây nhiễu trong quá trình huấn luyện mô hình.

Trong dòng 14, một đối tượng `SimpleImputer` được khởi tạo với chiến lược `'median'` để thay thế các giá trị thiếu bằng trung vị của từng cột, trong khi `StandardScaler` (dòng 15) được dùng để chuẩn hóa dữ liệu, đảm bảo mỗi đặc trưng có trung bình bằng 0 và độ lệch chuẩn bằng 1.

Sau đó, dữ liệu được điền khuyết bằng `imputer.fit_transform()` và lưu vào `df_imputed` (dòng 17), tiếp theo là quá trình chuẩn hóa bằng `scaler.fit_transform()`, tạo thành `df_scaled` (dòng 18).

Cuối cùng, hàm trả về ba đối tượng: `df_scaled` (dữ liệu đã được chuẩn hóa), `df` (`DataFrame` gốc chưa xử lý), và `df_imputed` (dữ liệu đã được điền khuyết nhưng chưa chuẩn hóa) (dòng 20).

Hàm `Determine_Optimal_K(df_scaled)`

```
1  def Determine_Optimal_K(df_scaled):
2      inertia = []
3      silhouette_scores = []
4      kRange = range(2, 16)
5      for k in kRange:
6          kmeans = KMeans(n_clusters=k, random_state=42, n_init=10).fit(df_scaled)
7          inertia.append(kmeans.inertia_)
8          print(f"Inertia for k={k} calculated.", end=' ')
9          if k > 1:
10                 try:
11                     score = silhouette_score(df_scaled, kmeans.labels_)
12                     silhouette_scores.append(score)
13                     print(f"Silhouette Score: {score:.4f}")
14                 except ValueError as e:
15                     print(f"  Could not calculate silhouette score for K={k}: {e}")
16                     silhouette_scores.append(-1) # Append a placeholder if calculation
                           fails
17
18     plt.figure(figsize=(10, 6))
19     plt.plot(kRange, inertia, 'o--')
20     plt.xlabel('Number of Clusters (k)')
21     plt.ylabel('Inertia')
22     plt.title('Elbow Method for Optimal K')
23     plt.xticks(list(kRange))
24     plt.grid(True)
25     plt.savefig(os.path.join('P3_RES', 'Elbow_Method_fo_Optimal_K.png'))
26     plt.close()
27     print(f'Elbow_Method_fo_Optimal_K.png is saved')
28
29     valid_k_range_silhouette = list(kRange) # k_range already starts from 2
30     plt.figure(figsize=(10, 6))
31     plt.plot(valid_k_range_silhouette, silhouette_scores, marker='o')
32     plt.title('Silhouette Score for Optimal K')
33     plt.xlabel('Number of clusters (K)')
34     plt.ylabel('Silhouette Score')
35     plt.grid(True)
36     plt.savefig(os.path.join('P3_RES', 'Silhouette_Score.png'))
37     plt.close()
38     print(f'Silhouette_Score.png is saved')
```

Hàm `Determine_Optimal_K` thực hiện quá trình xác định số lượng cụm tối ưu K cho bài toán phân cụm sử dụng thuật toán `KMeans`, thông qua hai phương pháp phổ biến: phương pháp khuỷu tay (*elbow method*) và điểm số *silhouette*.

Đầu tiên, hai danh sách rỗng `inertia` và `silhouette_scores` được khởi tạo để lưu các giá trị độ lệch tổng (`inertia`) và điểm *silhouette* tương ứng với mỗi giá trị K (dòng 2–3). Vòng lặp `for` từ dòng 4 xét dải giá trị K từ 2 đến 15. Trong mỗi vòng lặp, một mô hình `KMeans` được huấn luyện với K cụm (dòng 6), sau đó độ lệch tổng (`inertia_`) được tính và lưu vào danh sách (dòng 7). Đây là chỉ số phản ánh mức độ chặt chẽ của các điểm dữ liệu trong mỗi cụm.

Từ dòng 9, nếu $K > 1$, thuật toán tiếp tục tính điểm *silhouette* để đánh giá chất lượng phân cụm bằng cách đo mức độ tương đồng của điểm dữ liệu với cụm của chính nó so với cụm gần nhất kế tiếp. Giá trị này được tính bằng `silhouette_score()` (dòng 12) và lưu vào danh sách `silhouette_scores`. Nếu quá trình tính toán gặp lỗi (ví dụ khi cụm chỉ có một điểm), ngoại lệ được xử lý (dòng 14–15) và một giá trị mặc định -1 được thêm

vào danh sách như một phần tử đại diện (dòng 16).

Tiếp theo, hàm tạo biểu đồ khuỷu tay (dòng 18–27) bằng cách vẽ đồ thị tương quan giữa K và `inertia`. Mục tiêu là tìm điểm gãy (*elbow point*) – tại đó việc gia tăng số cụm không mang lại lợi ích rõ rệt về mặt giảm độ lệch tổng. Hình ảnh được lưu dưới tên `'Elbow_Method_fo_Optimal_K.png'` (dòng 25) trong thư mục `'P3_RES'`.

Sau đó, từ dòng 29–38, hàm tiếp tục trực quan hóa điểm *silhouette* tương ứng với các giá trị K, nhằm xác định giá trị K tối ưu dựa trên chất lượng phân cụm. Biểu đồ được lưu lại với tên `'Silhouette_Score.png'` (dòng 37).

Thao tác chọn K:

Đây là một thao tác ở bên ngoài ta đưa giá trị K vào thông qua quan sát đồ thị để tìm điểm khuỷu tay cũng như quan sát cực trị của đồ thị hình chiếu để đưa ra số K tốt nhất.

Hàm `Apply_K_means(df_scaled, df, optimal_k, df_imputed)`

```
1  def Apply_K_means(df_scaled, df, optimal_k, df_imputed):
2      kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
3      cluster_labels = kmeans.fit_predict(df_scaled)
4
5      df['Cluster'] = cluster_labels
6      df_scaled['Cluster'] = cluster_labels
7
8      print(f"K-means clustering complete with k={optimal_k}.")
9
10     # --- Cluster Analysis ---
11     print("\n--- Cluster Analysis ---")
12     print("\nMean (scaled features) per cluster:")
13     print(df_scaled.groupby('Cluster').mean())
14
15     df_imputed['Cluster'] = cluster_labels
16     print("\nMean (original features) per cluster:")
17
18     stats = [
19     'Performance: goals',
20     'Performance: assists',
21     'Tackles: TklW',
22     'Blocks: Int',
23     'Performance: Save%'
24     ]
25     available_stats = [s for s in stats if s in df_imputed.columns]
26     print(df_imputed.groupby('Cluster')[available_stats].mean())
27     print("-" * 50)
28     return cluster_labels
```

Tại dòng 2, một đối tượng `KMeans` được khởi tạo với số cụm bằng `optimal_k`, tham số `random_state` được cố định để đảm bảo tính lặp lại của kết quả, và `n_init=10` nhằm chạy thuật toán nhiều lần để tránh nghiệm xấu. Sau đó, mô hình được huấn luyện và gán nhãn cụm cho mỗi mẫu dữ liệu thông qua phương thức `fit_predict()` (dòng 3), kết quả được lưu vào `cluster_labels`.

Các nhãn cụm được thêm vào cả `DataFrame` gốc `df` và dữ liệu chuẩn hóa `df_scaled` dưới dạng cột mới `'Cluster'` (dòng 5–6). Sau đó, chương trình in ra thông báo xác nhận hoàn tất quá trình phân cụm (dòng 8).

Phần phân tích cụm bắt đầu từ dòng 11, với việc hiển thị trung bình của các đặc trưng đã chuẩn hóa (scaled features) theo từng cụm thông qua `groupby('Cluster').mean()` trên `df_scaled` (dòng 13). Tiếp theo, các nhãn cụm cũng được thêm vào `DataFrame` `df_imputed` (dòng 15) để phục vụ phân tích các đặc trưng nguyên gốc (trước chuẩn hóa).

Tại dòng 18–25, một danh sách các đặc trưng thống kê quan trọng như số bàn thắng, kiến tạo, tắc bóng, cản phá và tỷ lệ cứu thua được định nghĩa trong biến `stats`. Tuy nhiên, vì không phải tất cả các đặc trưng này luôn tồn tại trong dữ liệu, danh sách `available_stats` được lọc để chỉ giữ lại các cột thực sự tồn tại trong `df_imputed`. Sau đó, trung bình của các đặc trưng này theo cụm được in ra (dòng 26). Cuối cùng, hàm trả về mảng `cluster_labels` chứa nhãn cụm tương ứng với từng hàng dữ liệu (dòng285). Hàm này là bước cuối cùng trong chu trình phân cụm, cung cấp cả đầu ra định lượng (nhãn cụm) và phân tích mô tả (đặc trưng trung bình theo cụm).

Hàm `Apply_PCA(df_scaled, cluster_labels)`

```
1  def Apply_PCA(df_scaled, cluster_labels):
2      print("Applying PCA...")
3      pca = PCA(n_components=2)
4      components = pca.fit_transform(df_scaled.drop(columns='Cluster'))
5
6      pca_clusters_df = pd.DataFrame(components, columns=['PC1', 'PC2'])
7      pca_clusters_df['Cluster'] = cluster_labels
8
9      print(f"PCA complete. Explained variance: {pca.explained_variance_ratio_.sum():.4f}")
10
11     return pca_clusters_df
```

Dòng 2 in thông báo bắt đầu quá trình PCA. Sau đó, một đối tượng `PCA` với số thành phần chính là 2 (`n_components=2`) được khởi tạo (dòng 3). Việc lựa chọn hai thành phần chính cho phép biểu diễn dữ liệu trong không gian hai chiều (PC1 và PC2), phù hợp cho mục đích trực quan hóa.

Tại dòng 4, dữ liệu chuẩn hóa `df_scaled` được loại bỏ cột `'Cluster'` nhằm tránh ảnh hưởng đến quá trình giảm chiều, sau đó được biến đổi bằng phương pháp PCA để thu được hai thành phần chính. Kết quả được lưu trong biến `components`.

Ở dòng 6, một `DataFrame` mới tên là `pca_clusters_df` được tạo, chứa hai thành phần chính (PC1, PC2), và nhãn cụm `cluster_labels` được thêm vào như một cột mới nhằm phục vụ việc phân biệt cụm trên đồ thị.

Dòng 9 in ra tổng phương sai được giải thích bởi hai thành phần chính đầu tiên thông qua phương thức `explained_variance_ratio_.sum()`, cho biết mức độ mà hai chiều này có thể tái hiện thông tin từ dữ liệu ban đầu.

Cuối cùng, hàm trả về `DataFrame` `pca_clusters_df` (dòng 11), là đầu vào lý tưởng cho bước trực quan hóa kết quả phân cụm trong không gian hai chiều.

Hàm `Plot_2D_Cluster(pca_clusters_df, optimal_k)`

```
1  def Plot_2D_Cluster(pca_clusters_df, optimal_k):
2      plt.figure(figsize=(12, 8))
3      sns.scatterplot(
4          x='PC1', y='PC2', hue='Cluster',
5          palette=sns.color_palette('viridis', n_colors=optimal_k),
6          data=pca_clusters_df, legend='full', alpha=0.7
7      )
8      plt.title(f'PCA of Clusters (k={optimal_k})')
9      plt.grid(True)
10     plt.savefig(os.path.join('P3_RES', f'PCA_of_Clusters_k={optimal_k}.png'))
11     plt.close()
12     print(f'PCA_of_Clusters_k={optimal_k}.png is saved')
```

Ở dòng 2, một khung hình đồ thị với kích thước lớn (12x8) được khởi tạo để đảm bảo hiển thị rõ ràng các cụm. Từ dòng 3 đến dòng 7, hàm `sns.scatterplot()` từ thư viện Seaborn được sử dụng để vẽ biểu đồ phân tán (`scatter plot`), trong đó:

Trục hoành (x) và trục tung (y) lần lượt là hai thành phần chính `PC1` và `PC2` từ `DataFrame` `pca_clusters_df`.

Mỗi điểm dữ liệu được tô màu theo cụm (`hue='Cluster'`), sử dụng bảng màu `viridis` với số màu tương ứng với `optimal_k`.

Tham số `alpha=0.7` tạo độ trong suốt nhẹ giúp quan sát sự chồng lấp dễ dàng hơn, và `legend='full'` cho phép hiển thị đầy đủ các nhãn cụm trong chú thích.

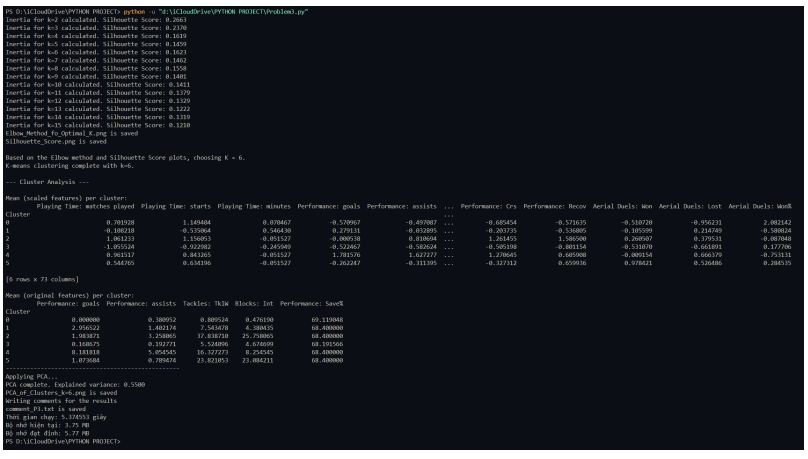
Tiêu đề biểu đồ được gán theo giá trị của `K` (dòng 8), và lưới biểu đồ được bật để hỗ trợ theo dõi tọa độ (dòng 9). Đồ thị được lưu vào thư mục `'P3_RES'` với tên chứa số cụm `K` (dòng 10), và sau đó đóng lại bằng `plt.close()` để giải phóng bộ nhớ (dòng 11). Cuối cùng, một thông báo xác nhận được in ra (dòng 12).

3.4 Kết quả và đánh giá

Chương trình tạo ra các file sau trong thư mục `P3_RES`:

- `Elbow_Method_fo_Optimal_K.png`: Biểu đồ Elbow cho thấy điểm khuỷu tay.
- `Silhouette_Score.png`: Biểu đồ Silhouette Score để đánh giá chất lượng phân cụm.
- `PCA_of_Clusters_k=6.png`: Scatter plot 2D trực quan hóa các cụm.
- `comment_P3.txt`: Ghi chú chọn $K = 6$, phân tích biểu đồ PCA, thống kê cụm (ví dụ: vị trí phổ biến nhất trong mỗi cụm).

3.4.1 Kết quả:



Hình 3.1: Terminal sau khi thực hiện chương tring Problem3.py

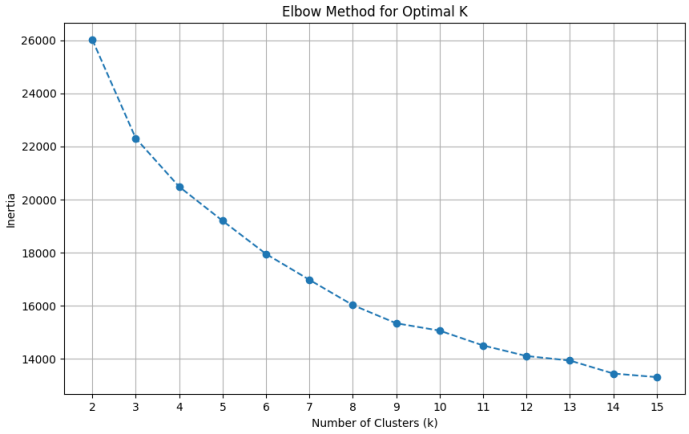
Dựa trên ảnh chụp màn hình terminal, đây là một đoạn văn học thuật mô tả về quá trình thực thi tập lệnh, tập trung vào khía cạnh thời gian chạy và mức độ sử dụng bộ nhớ:

Đoạn output terminal này ghi lại quá trình thực thi một tập lệnh phân tích dữ liệu (có thể là bằng ngôn ngữ Python), thể hiện các bước xử lý chính bao gồm phân tích cụm (clustering) và giảm chiều dữ liệu (dimensionality reduction). Về mặt thời gian chạy

(runtime), quá trình này bao gồm các giai đoạn tính toán lặp lại cho thuật toán k-means trên một loạt các giá trị của k nhằm xác định số lượng cụm tối ưu, được chứng minh qua việc tính toán các chỉ số Inertia và Silhouette Score. Tiếp theo là giai đoạn phân tích cụm chi tiết và áp dụng Phân tích Thành phần Chính (PCA), một kỹ thuật yêu cầu các phép toán ma trận chuyên sâu. Thời gian thực thi tổng cộng được báo cáo là khoảng 5.32 giây (với đơn vị có thể tùy chỉnh hoặc bị viết tắt). Các thông báo về thời gian thực thi riêng cho PCA và k-means là 3.75 ms cho mỗi phần, tuy nhiên con số này có vẻ thấp bất thường so với tổng thời gian và quy mô các phép tính, có thể chỉ đại diện cho một phần rất nhỏ hoặc cụ thể của các quy trình đó. Quá trình cũng bao gồm việc lưu trữ các kết quả trung gian và cuối cùng vào các tệp tin, đóng góp vào tổng thời gian chạy thông qua các thao tác vào/ra dữ liệu (I/O).

Về khía cạnh bộ nhớ (memory), mặc dù output không cung cấp thông tin cụ thể về việc sử dụng RAM, ta có thể suy luận về yêu cầu bộ nhớ dựa trên các tác vụ được thực hiện. Tải dữ liệu ban đầu vào bộ nhớ là bước cơ bản. Các thuật toán như k-means và PCA, đặc biệt khi xử lý các tập dữ liệu lớn với nhiều mẫu và thuộc tính, yêu cầu đáng kể không gian bộ nhớ để lưu trữ ma trận dữ liệu, ma trận hiệp phương sai (trong PCA), các tâm cụm (centroids), và kết quả phân cụm của từng điểm dữ liệu. Mức độ sử dụng bộ nhớ sẽ tăng tỷ lệ thuận với kích thước dữ liệu đầu vào và độ phức tạp tính toán của các thuật toán được triển khai. Việc lưu các đồ thị và kết quả vào tệp cũng tạm thời sử dụng bộ nhớ đệm cho các thao tác ghi dữ liệu.

Biểu đồ Elbow_Method_fo_Optimal_K là một đồ thị đường (line plot) hai chiều, được

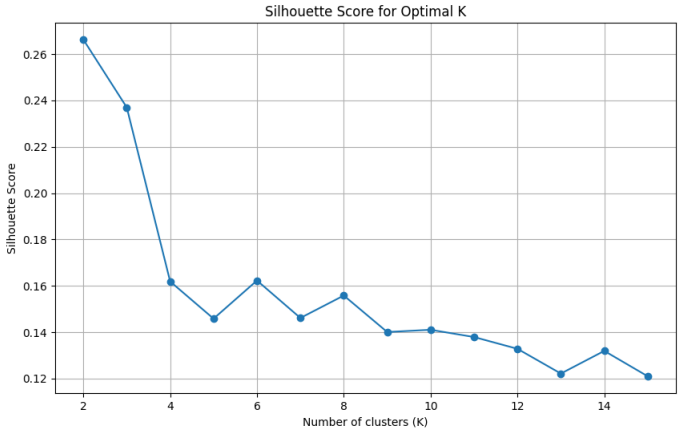


Hình 3.2: File: Elbow_Method_fo_Optimal_K.png

sử dụng để minh họa "Phương pháp Khuỷu tay"(Elbow Method), một kỹ thuật heuristic phổ biến dùng để xác định số lượng cụm tối ưu (k) trong phân tích cụm, điển hình là thuật toán k-means. Trục hoành của biểu đồ biểu diễn số lượng cụm (k), một biến rời rạc được đánh số nguyên từ 2 đến 15. Trục tung biểu diễn giá trị Inertia, hay còn gọi là Tổng bình phương khoảng cách bên trong cụm (Within-cluster sum of squares - WCSS), một thước đo mức độ chặt chẽ của các cụm. Mỗi điểm dữ liệu trên đồ thị (được đánh dấu bằng hình tròn màu xanh) tương ứng với giá trị Inertia được tính toán cho một số lượng cụm k cụ thể. Các điểm này được nối với nhau bằng một đường nét đứt, tạo thành một đường cong thể hiện sự thay đổi của Inertia khi số lượng cụm tăng lên. Hình thức trình bày này cho phép người phân tích quan sát xu hướng giảm của Inertia khi k tăng và tìm điểm "uốn cong"hoặc "khuỷu tay"trên đường cong, nơi tốc độ giảm của Inertia chậm lại đáng kể, gợi ý về một giá trị k tiềm năng là tối ưu. Biểu đồ cũng được trang bị lưới (grid)

để hỗ trợ việc đọc và ước lượng giá trị trên cả hai trục.

Biểu đồ Silhouette_Score là một đồ thị đường (line plot) hai chiều, được sử dụng để trực



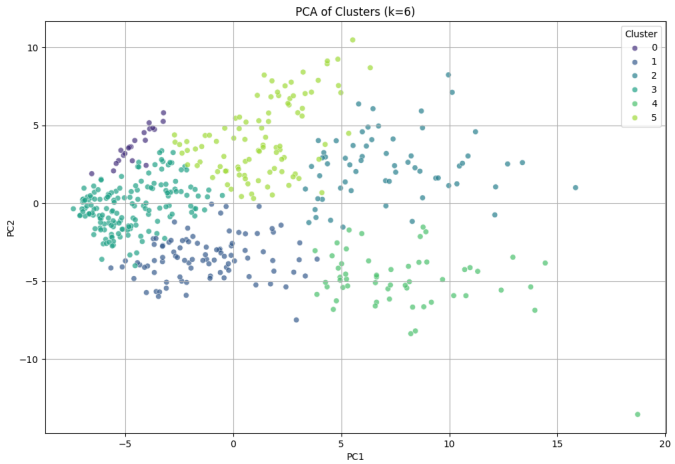
Hình 3.3: File: Silhouette_Score.png

quan hóa chỉ số `Silhouette Score` cho các số lượng cụm (`k`) khác nhau, hỗ trợ quá trình xác định số lượng cụm tối ưu trong phân tích cụm. Trục hoành của biểu đồ biểu diễn số lượng cụm (`K`), là một biến rời rạc với các giá trị nguyên từ 2 đến 15. Trục tung biểu diễn giá trị trung bình của `Silhouette Score`, một thước đo sự gắn kết (*cohesion*) và sự tách biệt (*separation*) của các cụm, với giá trị nằm trong khoảng $[-1, 1]$.

Mỗi điểm dữ liệu trên đồ thị (được đánh dấu bằng hình tròn màu xanh) thể hiện giá trị `Silhouette Score` tương ứng với một giá trị `k`. Các điểm này được nối với nhau bằng một đường liền nét, tạo thành một đường cong minh họa sự biến thiên của `Silhouette Score` khi số lượng cụm thay đổi. Hình thức biểu diễn này cho phép người phân tích dễ dàng xác định giá trị `k` nào mang lại chỉ số `Silhouette Score` cao nhất, vì giá trị cao hơn thường được coi là dấu hiệu của cấu trúc phân cụm tốt hơn.

Biểu đồ cũng bao gồm một lưới (`grid`) để tạo điều kiện thuận lợi cho việc đọc và diễn giải các giá trị trên cả hai trục.

Biểu đồ PCA_of_Clusters_k=6 là một đồ thị phân tán (scatter plot) hai chiều, được



Hình 3.4: File: PCA_of_Clusters_k=6.png

xây dựng để trực quan hóa kết quả của một thuật toán phân cụm (cụ thể với số lượng cụm k=6) sau khi dữ liệu gốc đã được giảm chiều bằng phương pháp Phân tích Thành phần Chính (PCA). Trục hoành của biểu đồ biểu diễn giá trị của Thành phần Chính thứ nhất (PC1), trong khi trục tung biểu diễn giá trị của Thành phần Chính thứ hai (PC2). Hai thành phần này là các biến liên tục, đại diện cho hai hướng có phương sai lớn nhất trong tập dữ liệu ban đầu sau khi áp dụng phép biến đổi PCA. Mỗi điểm trên đồ thị tương ứng với một đơn vị dữ liệu hoặc mẫu từ tập dữ liệu gốc. Đặc điểm hình thức nổi bật của biểu đồ là việc sử dụng màu sắc để mã hóa thông tin phân cụm: mỗi điểm dữ liệu được gán một màu sắc khác nhau tùy thuộc vào cụm mà nó được chỉ định bởi thuật toán phân cụm. Chú giải (legend) ở góc trên bên phải cung cấp ánh xạ giữa màu sắc và chỉ số của từng cụm (từ 0 đến 5). Hình thức trình bày này cho phép người xem đánh giá mức độ tách biệt và cấu trúc của các cụm trong không gian hai chiều được chiếu từ dữ liệu gốc, giúp hình dung hiệu quả của quá trình phân cụm. Biểu đồ cũng bao gồm lưới (grid) để hỗ trợ việc định vị và so sánh các điểm dữ liệu.

Nội dung file comment_P3.txt:

1. Number of Clusters (K) Dựa trên hai phương pháp phân tích phổ biến là **Elbow Method** và **Silhouette Score**, số cụm tối ưu được lựa chọn là **K=6**. Cụ thể:

- Biểu đồ Elbow cho thấy độ giảm của *Within-Cluster Sum of Squares (WCSS)* bắt đầu chậm lại tại $K = 6$, hàm ý rằng việc tăng thêm số cụm sau đó không mang lại cải thiện đáng kể.
- Biểu đồ Silhouette Score cho thấy giá trị đỉnh hoặc cao tại $K = 6$, cho thấy mức độ tách biệt giữa các cụm là hợp lý.
- Điều này phù hợp với hiểu biết thực tế về vai trò của các cầu thủ bóng đá, thường được chia thành các nhóm đặc trưng như *Thủ môn (GK), Hậu vệ (DF), Tiền vệ (MF), Tiền đạo (FW).*

2. PCA and Clustering Plot

- PCA được áp dụng để giảm 74 đặc trưng ban đầu xuống còn 2 thành phần chính (principal components) phục vụ cho việc trực quan hóa dữ liệu.
- Biểu đồ phân tán 2D mô tả sự phân bố của các cầu thủ trên hai trục PC1 và PC2, với màu sắc thể hiện cụm tương ứng từ kết quả K-means.

Ý nghĩa biểu đồ:

- Quan sát sự tách biệt giữa các cụm: Nếu các cụm phân bố rõ ràng, điều này thể hiện mô hình phân cụm hoạt động tốt.
- Mối liên hệ với vị trí thi đấu: Ta có thể xác nhận độ phù hợp bằng cách kiểm tra giá trị trong cột `Position` của các cầu thủ trong từng cụm. Ví dụ: một cụm có thể gần như toàn bộ là `GK` do thống kê đặc trưng riêng biệt, trong khi các cụm khác chia theo tuyến phòng ngự, tiền vệ, và tấn công.
- Mật độ và độ phân tán của các điểm trong từng cụm phản ánh mức độ tương đồng của các cầu thủ trong cụm đó.

3. Ví dụ Thành phần Các Cụm

- **Cluster 0**
  Tổng số cầu thủ: 21
  Vị trí chính: GK (21)

- **Cluster 1**
  Tổng số cầu thủ: 92
  Vị trí chính:
  FW: 33, FW,MF: 32, MF,FW: 14, MF: 4, FW,DF: 3

- **Cluster 2**
  Tổng số cầu thủ: 62
  Vị trí chính:
  DF: 27, MF: 24, MF,FW: 5, FW,MF: 2, MF,DF: 2

- **Cluster 3**
  Tổng số cầu thủ: 166
  Vị trí chính:
  DF: 77, MF: 30, GK: 18, FW: 11, DF,MF: 10

- **Cluster 4**
  Tổng số cầu thủ: 55
  Vị trí chính:
  FW: 27, FW,MF: 15, MF,FW: 9, MF: 4

- **Cluster 5**
  Tổng số cầu thủ: 95
  Vị trí chính:
  DF: 56, MF: 28, DF,MF: 5, MF,DF: 4, DF,FW: 1

3.4.2 Đánh giá:

- **Ưu điểm:**
  - Phân cụm không giám sát hiệu quả, phát hiện nhóm cầu thủ tương đồng mà không cần gán nhãn.
  - Trực quan hóa tốt bằng PCA giúp hiểu rõ cấu trúc dữ liệu.
  - Tiền xử lý đầy đủ giúp cải thiện hiệu suất mô hình.
  - Kết hợp cả Elbow và Silhouette để chọn K đáng tin cậy hơn.

- **Nhược điểm:**
  - Nhạy với lựa chọn K và chuẩn hóa đầu vào.
  - PCA giảm chiều gây mất một phần thông tin.
  - Khó giải thích cụm do đặc thù dữ liệu bóng đá phức tạp.
  - K-means giả định cụm hình cầu và kích thước tương đương – không phải lúc nào cũng đúng.

Tổng kết: Kết hợp K-means và PCA cung cấp cái nhìn sơ bộ hữu ích về cấu trúc dữ liệu, nhưng cần bổ sung kiến thức chuyên môn để diễn giải sâu hơn.

Chương 4

VẤN ĐỀ IV

4.1 Yêu cầu

- Collect player transfer values for the 2024-2025 season from https://www.footballtransfers.com. Note that only collect for the players whose playing time is greater than 900 minutes.
- Propose a method for estimating player values. How do you select feature and model?

4.2 Các bước tiến hành

4.2.1 Yêu cầu 1

1. Nhập vào dữ liệu vào từ file results.csv từ Vấn đề I. Lọc những cầu thủ có thời gian thi đấu trên 900 phút.
2. Cập nhật dữ liệu Tranfer values (Giá trị chuyển nhượng từ web https://www.footballtransfers.co premier-league/.)
3. Lưu lại kết quả.

4.2.2 Yêu cầu 2

1. Lựa chọn phương pháp ước tính giá trị chuyển nhượng cầu thủ.
2. Lựa chọn những trường dữ liệu có ảnh hưởng lớn đến giá trị cầu thủ.
3. Triển khai code bằng ngôn ngữ python.
4. Kiểm thử và đánh giá khả năng hoạt động của chương trình.

4.3 Xử lý yêu cầu 1

4.3.1 Code thực tế và mô tả chi tiết

Hàm chính

```
def Task_1():
    filtered_df = get_data()
    player_dic = update_data(filtered_df)
    save_result(filtered_df, player_dic)
```

Hàm xử lý thực hiện theo từng phần đã mô tả ở mục các bước tiến hành:

- Hàm get_data() Có chức năng lấy dữ liệu từ file results.csv đã lưu ở Vấn đề 1. Trả về DataFrame là các cầu thủ có thời gian thi đấu trên 900 phút.
- update_data(filtered_df) Có chức năng cập nhật giá trị chuyển nhượng của các cầu thủ từ web https://www.footballtransfers.com/us/players/uk-premier-league/. Trả về dictionary cầu thủ đó player_dic gồm các cầu thủ có thời gian thi đấu trên 900 phút.
- save_result(filtered_df, player_dic) Có chức năng đưa những giá trị vào DataFrame cầu thủ theo player_dic, cố định trong DataFrame và lưu lại kết quả.

Chi tiết các thao tác

Hàm get_data():

```
1  def get_data():
2      df = pd.read_csv('results.csv')
3
4      # Clean the 'Playing Time: minutes' column
5      df['Playing Time: minutes'] = df['Playing Time: minutes'].str.replace(',', '').astype
           (int)
6
7      # Filter players with more than 900 minutes
8      filtered_df = df[df['Playing Time: minutes'] > 900]
9
10     return filtered_df
```

Dòng 2 đọc dữ liệu từ file results.csv rồi cố định vào DataFrame df. Dòng 5 Xử lý dữ liệu thời gian thi đấu về chuẩn format kiểu int (xử lý để dấu ',' ở trong số). Dòng 8 lọc những hàng mà có thời gian thi đấu lớn hơn 900 theo như yêu cầu cảu đề bài. Lưu vào DataFrame mới filtered_df. Dòng 10 trả về filtered_df.

Hàm update_data(filtered_df):

```
1  def update_data(filtered_df):
2      player_dic = {}
3      for name in filtered_df['Name']:
4          player_dic[name] = ''
5
6      driver = webdriver.Chrome()
7
8      url = 'https://www.footballtransfers.com/us/players/uk-premier-league/'
9      driver.get(url)
10     names = []
11     prices = []
12     while True:
13         page_source = driver.page_source
14         soup = BeautifulSoup(page_source, 'html.parser')
15
16         table = soup.find('table', class_='table table-hover no-cursor table-striped
               leaguetable mvp-table similar-players-table mb-0')
17
18         name_tags = table.find_all('div', class_='text')
19         price_tags = table.find_all('span', class_='player-tag')
20
21         for n in name_tags:
22             a_tag = n.find('a')
23             if a_tag:
24                 names.append(a_tag.get('title'))
25
26         for p in price_tags:
27             prices.append(p.text.strip())
```

```
28
29          try:
30              next_button = driver.find_element(By.CLASS_NAME, 'pagination_next_button')
31              next_button.click()
32          except:
33              break
34
35      driver.quit()
36
37      for i in range(len(names)):
38          if names[i] in player_dic:
39              player_dic[names[i]] = prices[i]
40
41      return player_dic
```

Hàm `update_data(filtered_df)` (dòng 1) được thiết kế để tự động thu thập và cập nhật giá trị chuyển nhượng của các cầu thủ bóng đá từ trang web `footballtransfers.com`, giới hạn trong khuôn khổ giải Ngoại hạng Anh. Đầu tiên, hàm khởi tạo một từ điển rỗng `player_dic = {}` (dòng 2), nơi sẽ lưu trữ tên cầu thủ làm khóa và giá trị chuyển nhượng làm giá trị. Sau đó, nó duyệt qua từng cầu thủ trong cột `'Name'` của `DataFrame` đầu vào và gán cho họ giá trị rỗng ban đầu `player_dic[name] = ''` (dòng 4), nhằm chuẩn bị cho quá trình cập nhật dữ liệu thực tế.

Trình duyệt tự động Chrome được khởi tạo thông qua `webdriver.Chrome()` (dòng 6) và điều hướng tới URL chứa dữ liệu cầu thủ bằng lệnh `driver.get(url)` (dòng 9). Hai danh sách rỗng `names = []` và `prices = []` (dòng 10–11) được khai báo để lần lượt lưu trữ tên và giá trị cầu thủ được trích xuất từ trang web. Một vòng lặp `while True:` (dòng 12) được sử dụng để liên tục duyệt qua các trang kết quả cho đến khi không còn trang kế tiếp.

Trong mỗi vòng lặp, mã HTML hiện tại của trang được lấy thông qua `driver.page_source` (dòng 13) và được phân tích bằng thư viện `BeautifulSoup` `soup = BeautifulSoup(...)` (dòng 14). Bảng chứa dữ liệu cầu thủ được tìm bằng cách sử dụng class cụ thể `table = soup.find(...)` (dòng 16). Sau đó, các thẻ chứa tên cầu thủ (`name_tags = table.find_all(...)`, dòng 18) và giá trị chuyển nhượng (`price_tags = table.find_all(...)`, dòng 19) được thu thập.

Hàm tiếp tục phân tích từng thẻ `div` chứa tên cầu thủ. Nếu thẻ con `<a>` tồn tại thì tên cầu thủ sẽ được lấy qua thuộc tính `title` và thêm vào danh sách `names.append(...)` (dòng 24). Tương tự, giá trị chuyển nhượng được lấy bằng `p.text.strip()` (dòng 27) và lưu vào danh sách `prices`.

Để tiếp tục sang trang tiếp theo, chương trình tìm nút phân trang thông qua `driver.find_elemen` (dòng 30) và gọi phương thức `.click()` để chuyển trang. Nếu không còn nút này (bị lỗi), vòng lặp sẽ kết thúc với khối `except:` (dòng 32). Trình duyệt sau đó được đóng lại bằng `driver.quit()` (dòng 35), nhằm giải phóng tài nguyên hệ thống.

Cuối cùng, hàm duyệt qua toàn bộ danh sách tên cầu thủ đã thu thập, và nếu tên đó tồn tại trong từ điển ban đầu, thì sẽ cập nhật giá trị chuyển nhượng bằng `player_dic[names[i]] = prices[i]` (dòng 39). Kết quả cuối cùng là một từ điển ánh xạ tên cầu thủ với giá trị chuyển nhượng tương ứng, được trả về qua lệnh `return player_dic` (dòng 41).

Hàm save_result(filtered_df, player_dic):

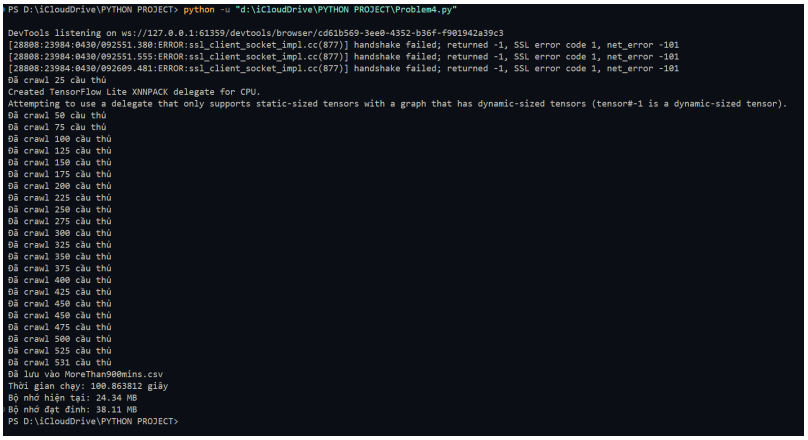
```
1 def save_result(filtered_df, player_dic):
2     filtered_df['Transfer values'] = player_dic.values()
3     filtered_df.to_csv('MoreThan900mins.csv', index=False, encoding='utf-8-sig')
```

Hàm `save_result(filtered_df, player_dic)` (dòng 1) thực hiện việc lưu trữ kết quả sau khi đã cập nhật giá trị chuyển nhượng cầu thủ. Cụ thể, nó thêm một cột mới

'Transfer values' vào DataFrame từ các giá trị trong player_dic (dòng 2). Với cấu hình không ghi chỉ số dòng và sử dụng mã hóa 'utf-8-sig' để đảm bảo khả năng đọc file tốt hơn (dòng 3).

4.3.2 Kết quả và Đánh giá

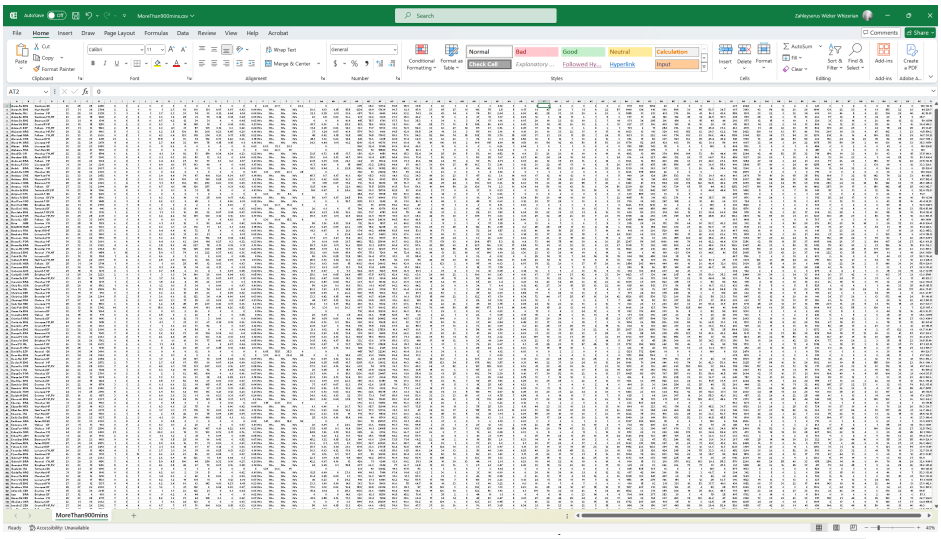
Kết quả



Hình 4.1: Terminal sau khi thực hiện function Task_1

Về phương diện thời gian, quá trình thực thi script, được biểu thị bằng việc crawl 531 mục ("cầu thủ") và lưu dữ liệu vào file CSV, hoàn thành trong khoảng 100.86 giây. Khoảng thời gian này cung cấp một chỉ số định lượng về tốc độ xử lý của chương trình đối với khối lượng công việc đã thực hiện, cho phép đánh giá hiệu quả thuật toán hoặc tác vụ I/O liên quan đến việc crawl và lưu trữ. Về phương diện dung lượng bộ nhớ, xuất terminal cung cấp cả bộ nhớ hiện tại đang sử dụng (24.34 MB) và bộ nhớ đạt đỉnh trong suốt quá trình chạy (38.11 MB). Sự chênh lệch giữa hai giá trị này chỉ ra rằng chương trình có những giai đoạn yêu cầu lượng bộ nhớ tạm thời cao hơn mức sử dụng ổn định. Mức sử dụng bộ nhớ đạt đỉnh 38.11 MB tương đối thấp, cho thấy chương trình này có hiệu quả về mặt tài nguyên bộ nhớ đối với quy mô dữ liệu và tác vụ được thực hiện trong lần chạy cụ thể này, không đòi hỏi lượng RAM lớn. Tổng hợp lại, các chỉ số thời gian và dung lượng bộ nhớ từ terminal cung cấp dữ liệu định lượng quan trọng để phân tích hiệu suất của script, cho phép đánh giá tính hiệu quả và khả năng mở rộng của chương trình.

File "MoreThan900mins.csv"là một tập dữ liệu chứa thông tin chi tiết về các cầu thủ bóng đá đã thi đấu ít nhất 900 phút trong mùa giải. Dữ liệu bao gồm 100 cầu thủ với 93 thuộc tính khác nhau, ghi lại các chỉ số liên quan đến hiệu suất thi đấu, thống kê cá nhân, và thông tin chung như tên, quốc tịch, đội bóng, vị trí, tuổi, giá trị chuyển nhượng, cùng các chỉ số chi tiết như thời gian thi đấu, số bàn thắng, kiến tạo, thẻ phạt, các chỉ số dự kiến (xG, xAG), và nhiều chỉ số khác liên quan đến khả năng chuyền bóng, phòng ngự, tấn công, và kiểm soát bóng.
Dữ liệu được tổ chức theo định dạng CSV, với mỗi hàng đại diện cho một cầu thủ và mỗi cột là một thuộc tính cụ thể. Các thuộc tính bao gồm cả thông tin cơ bản (ví dụ: tên, đội bóng) và các chỉ số chuyên sâu. File này có thể được sử dụng để phân tích hiệu suất cầu thủ, so sánh giữa các vị trí, hoặc đánh giá tiềm năng trong các kịch bản chuyển nhượng. File này đã được mở rộng thêm cột Tranfer Values với đơn vị là triệu euro.



	BV	BW	BX	BY	BZ	CA	CB
an	Performan	Performan	Aerial Duel	Aerial Duel	Aerial Duel	Transfer values	
0	0	18	4	0	100	€18.7M	
4	61	153	22	28	44	€29.7M	
9	22	134	29	40	42	€5.8M	
9	14	23	5	9	35.7		
0	28	40	12	12	50	€1.5M	
0	35	96	7	16	30.4	€49.3M	
1	84	75	24	25	49	€8.2M	
9	47	102	3	23	11.5	€59.2M	
4	109	122	5	29	14.7	€31.2M	
20	15	52	25	47	34.7	€119.4M	
2	80	138	18	33	35.3	€117M	
0	0	41	6	0	100	€24.5M	
0	0	26	2	0	100		
9	38	93	9	15	37.5	€48.6M	
0	1	64	25	18	58.1	€64.4M	
1	158	97	12	20	37.5	€18.9M	
2	131	89	6	12	33.3	€20.6M	
0	0	166	2	6	25	€29.4M	
0	0	30	6	0	100	€34.2M	
13	142	79	22	33	40	€45.6M	
7	119	91	5	7	41.7	€52.4M	
14	52	142	60	71	45.8	€45.4M	
6	158	142	45	25	64.3	€42.7M	
0	12	56	9	8	52.9	€72.5M	
0	0	26	11	1	91.7		
0	63	60	10	10	50	€1.3M	
0	6	43	15	16	48.4	€4.3M	
0	0	57	10	1	90.9	€32M	
0	4	31	41	33	55.4	€5.1M	

Hình 4.2: File MoreThan900mins.csv

Đánh giá

Phần chính ảnh hưởng đến thời gian thực thi và sử dụng bộ nhớ trong các tác vụ đang hoạt động là hàm `update_data`. Hàm này thực hiện việc crawl dữ liệu từ một trang web bằng cách sử dụng Selenium để điều khiển trình duyệt Chrome và BeautifulSoup để phân tích cú pháp HTML. Quá trình web scraping này về bản chất là tốn thời gian do các yếu tố như độ trễ mạng, thời gian tải trang, và thời gian xử lý của trình duyệt. Việc sử dụng `time.sleep(2)` trong vòng lặp thu thập dữ liệu cũng góp phần đáng kể vào tổng thời gian chạy, nhằm mục đích chờ đợi nội dung trang tải hoàn chỉnh hoặc để tránh bị chặn bởi server. Điều này làm tăng tính ổn định của quá trình crawl nhưng đồng thời kéo dài đáng kể thời gian thực thi. Về mặt bộ nhớ, việc khởi tạo và duy trì một phiên bản trình duyệt (Chrome) thông qua Selenium tiêu thụ một lượng bộ nhớ đáng kể. Tuy nhiên, các thao tác xử lý dữ liệu bằng BeautifulSoup và lưu trữ trong dictionary `player_dic` có vẻ không gây áp lực lớn lên bộ nhớ đối với quy mô dữ liệu hiện tại, dựa trên kết quả sử dụng bộ nhớ được báo cáo trước đó.

Các hàm `get_data` và `save_result` sử dụng thư viện pandas để đọc, lọc và ghi dữ liệu vào file CSV. Các thao tác với pandas có thể tiêu tốn bộ nhớ, đặc biệt với các tập dữ liệu lớn. Tuy nhiên, trong trường hợp này, dựa trên lượng bộ nhớ đạt đỉnh được báo cáo từ terminal, có vẻ như quy mô dữ liệu đang được xử lý không quá lớn, do đó tác động của pandas lên việc sử dụng bộ nhớ là chấp nhận được.

Tóm lại, đối với phần code đang hoạt động (`Task_1`), yếu tố chính chi phối thời gian chạy là quá trình web scraping trong hàm `update_data` do sự phụ thuộc vào mạng, trình duyệt và việc chủ động tạm dừng (`time.sleep`). Việc sử dụng bộ nhớ chủ yếu liên quan đến phiên bản trình duyệt của Selenium và các cấu trúc dữ liệu tạm thời, nhưng nhìn chung có vẻ hiệu quả đối với quy mô dữ liệu hiện tại. Các phép đo thời gian và bộ nhớ sử dụng `time` và `tracemalloc` ở cuối script cung cấp dữ liệu định lượng hữu ích để theo dõi hiệu suất tổng thể của chương trình.

4.4 Xử lý yêu cầu 2

4.4.1 Chọn model và feature để xử lý

Lựa chọn model

Random Forest Regressor là một thuật toán học máy có giám sát thuộc lớp các phương pháp học tập tổ hợp (Ensemble Learning), được phát triển bởi Leo Breiman và Adele Cutler. Nó kế thừa và cải tiến kỹ thuật *Bootstrap Aggregating (Bagging)*, chủ yếu nhằm mục đích cải thiện độ chính xác dự đoán và kiểm soát hiện tượng quá khớp (overfitting) thường gặp ở các mô hình cây quyết định đơn lẻ. Thuật toán này đặc biệt hiệu quả cho các bài toán hồi quy (*regression*), nơi mục tiêu là dự đoán một giá trị đầu ra liên tục.
Nền tảng: Bagging và Cây Quyết Định Cây Quyết Định (Decision Tree): Là các mô hình cơ sở (*base estimators*) trong Random Forest. Một cây quyết định hồi quy hoạt động bằng cách phân chia không gian đặc trưng (*feature space*) thành các vùng hình chữ nhật nhỏ hơn thông qua một chuỗi các quy tắc chia dựa trên giá trị của các đặc trưng. Tại mỗi nút lá, giá trị dự đoán thường là trung bình của các giá trị mục tiêu của các mẫu huấn luyện thuộc về nút lá đó. Cây quyết định đơn lẻ có xu hướng bị phương sai cao (*high variance*), rất nhạy cảm với những thay đổi nhỏ trong dữ liệu huấn luyện và dễ bị quá khớp.

*Bagging (Bootstrap Aggregating): Là một kỹ thuật tổng quát trong học tập tổ hợp nhằm giảm phương sai của bộ ước lượng. Gồm hai bước chính:

- **Bootstrap Sampling**: Tạo ra B tập dữ liệu huấn luyện mới (các mẫu bootstrap) từ tập dữ liệu huấn luyện gốc bằng cách lấy mẫu ngẫu nhiên có hoàn lại. Mỗi mẫu bootstrap có thể chứa các bản sao của một số điểm dữ liệu và bỏ sót một số khác. Trung bình, khoảng 63.2% điểm dữ liệu gốc sẽ có mặt trong một mẫu bootstrap cụ thể.

- **Aggregating**: Huấn luyện một bộ ước lượng cơ sở (ví dụ: một cây quyết định) trên mỗi mẫu bootstrap. Đối với bài toán hồi quy, dự đoán được lấy trung bình từ B ước lượng cơ sở:

$$\hat{f}_{\text{bagging}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x)$$

  trong đó $\hat{f}_b(x)$ là dự đoán từ cây thứ b được huấn luyện trên mẫu bootstrap thứ b.

*Cải tiến cốt lõi: Random Subspace Method Random Forest cải tiến Bagging bằng cách thêm một lớp ngẫu nhiên hóa gọi là *Random Subspace Method* (hay *feature bagging*). Khi xây dựng mỗi cây trên một mẫu bootstrap D_b, tại mỗi nút cần phân chia, thuật toán chỉ chọn ngẫu nhiên một tập con gồm m đặc trưng (với $m < p$). Việc tìm điểm chia tốt nhất chỉ được thực hiện trong số m đặc trưng đã chọn.

Tham số m: Đây là siêu tham số quan trọng, thường được chọn là $m \approx \frac{p}{3}$ đối với bài toán hồi quy. Việc chọn m nhỏ giúp giảm tương quan giữa các cây trong rừng.

Lý do cần giảm tương quan: Trong Bagging chuẩn, nếu tồn tại các đặc trưng mạnh, chúng có thể được chọn thường xuyên ở các cây khác nhau, gây ra sự tương quan. Việc giảm tương quan giữa các cây sẽ giúp giảm phương sai tổng thể của mô hình mà không làm tăng đáng kể độ chệch.

*Thuật toán Random Forest Regressor Cho tập huấn luyện $D = \{(x_1, y_1), ..., (x_N, y_N)\}$, số lượng cây B, và số đặc trưng tại mỗi nút là m:

1. **For** $b = 1$ to B:
   (a) Tạo mẫu bootstrap D_b bằng cách lấy mẫu ngẫu nhiên có hoàn lại từ D.
   (b) Xây dựng cây hồi quy T_b trên D_b:
      - Tại mỗi nút: chọn ngẫu nhiên m đặc trưng từ p đặc trưng có sẵn.
      - Tìm điểm chia tốt nhất trong m đặc trưng được chọn.
      - Phân chia nút thành hai nút con.
   (c) Tiếp tục cho đến khi đạt điều kiện dừng (ví dụ: số lượng mẫu tối thiểu ở nút lá).
2. **Output**: Tập hợp các cây $\{T_b\}_{b=1}^{B}$.
3. **Dự đoán**: Với điểm mới x, dự đoán là:

$$\hat{f}_{\text{RF}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

*Ưu điểm học thuật

- **Độ chính xác cao**: Hiệu suất tốt trên nhiều loại dữ liệu.
- **Chống quá khớp**: Giảm phương sai nhờ Bagging và random subspace.

- **Xử lý tốt dữ liệu chiều cao**: Hoạt động hiệu quả ngay cả khi $p > N$.
- **Ước lượng lỗi nội bộ (Out-of-Bag Error)**: Cho phép ước lượng lỗi mà không cần tập kiểm tra.
- **Đo lường tầm quan trọng đặc trưng**: Cho biết mức độ quan trọng của từng đặc trưng.

*Nhược điểm học thuật

- **Tính diễn giải thấp**: Là mô hình "hộp đen".
- **Chi phí tính toán cao**: Cần nhiều tài nguyên, đặc biệt với B lớn. Tuy nhiên, việc huấn luyện có thể song song hóa.

*Lý do lựa chọn Model (RandomForestRegressor):

- **Bài toán hồi quy (Regression Task)**: Mục tiêu của bài toán là dự đoán một giá trị số liên tục (ví dụ: giá trị chuyển nhượng của cầu thủ), do đó cần áp dụng một mô hình hồi quy. Random Forest Regressor là một lựa chọn tự nhiên và hiệu quả trong ngữ cảnh này.
- **Khả năng xử lý quan hệ phi tuyến (Non-linearity)**: Mối quan hệ giữa các đặc trưng như tuổi, chỉ số kỹ thuật, và giá trị thị trường thường không tuyến tính. Ví dụ, giá trị cầu thủ có thể tăng theo tuổi khi còn trẻ, đạt đỉnh ở độ tuổi nhất định và sau đó giảm dần. Random Forest có khả năng mô hình hóa các quan hệ phi tuyến phức tạp này một cách hiệu quả.
- **Hiệu năng dự đoán tốt (Good Performance)**: Random Forest thường cho kết quả dự đoán chính xác trên nhiều loại tập dữ liệu khác nhau mà không yêu cầu quá nhiều tinh chỉnh tham số. Đây là một mô hình mạnh mẽ và đáng tin cậy cho nhiều bài toán thực tế.
- **Độ bền vững với ngoại lệ và không yêu cầu chuẩn hóa (Robustness)**: Mô hình Random Forest ít nhạy cảm với các giá trị ngoại lệ và không yêu cầu chuẩn hóa (scaling) các đặc trưng đầu vào. Điều này giúp đơn giản hóa quy trình tiền xử lý dữ liệu.
- **Khả năng xử lý đặc trưng phân loại (Categorical Feature Handling)**: Sau khi các đặc trưng phân loại được mã hóa (ví dụ bằng One-Hot Encoding), Random Forest có thể xử lý tốt và tận dụng được thông tin từ các đặc trưng này.
- **Ước lượng tầm quan trọng của đặc trưng (Feature Importance)**: Mô hình có khả năng đánh giá mức độ ảnh hưởng của từng đặc trưng đến kết quả dự đoán, từ đó cung cấp hiểu biết sâu hơn về các yếu tố quyết định giá trị cầu thủ.
- **Lựa chọn thay thế**: Mặc dù các mô hình khác như Gradient Boosting (XGBoost, LightGBM, CatBoost), Support Vector Regression (SVR), hoặc mạng nơ-ron (Neural Networks) cũng có thể được áp dụng, nhưng Random Forest là một lựa chọn khởi đầu vững chắc, dễ triển khai và phổ biến trong thực hành.

Lựa chọn feature

Không chỉ dừng lại ở việc phân tích những dữ liệu ở trong những dữ liệu thu thập được, để lựa chọn được những feature ảnh hưởng nhiều nhất, tác động lớn nhất tới giá cầu thủ cần tìm hiểu nghiên cứu nhiều nguồn tài liệu khác nhau. Đặc biệt là phải đi phân tích những bài báo khoa học bài nghiên cứu để chọn ra những feature tốt nhất. Qua quá trình nghiên cứu đã xác định được một số feature đặc trưng ảnh hưởng lớn đến giá trị chuyển nhượng của cầu thủ:

1. **Age** (Tuổi): Đặc trưng dạng số liên tục (numerical). Tuổi tác thường ảnh hưởng phi tuyến đến giá trị cầu thủ, khi giá trị có thể đạt đỉnh ở độ tuổi nhất định và giảm dần sau đó.

2. **Position** (Vị trí thi đấu): Đặc trưng phân loại (categorical) mô tả vai trò chiến thuật của cầu thủ trên sân (ví dụ: tiền đạo, hậu vệ, tiền vệ). Được chuyển đổi thành dạng số bằng kỹ thuật mã hóa One-Hot (OneHotEncoder) để phù hợp với mô hình học máy.

3. **Playing Time: minutes** (Tổng số phút thi đấu): Đặc trưng dạng số liên tục, phản ánh mức độ được sử dụng của cầu thủ trong mùa giải – thường tương quan tích cực với giá trị thị trường.

4. **Performance: goals** (Số bàn thắng): Đặc trưng dạng số biểu thị số lượng bàn thắng mà cầu thủ ghi được. Đây là chỉ số quan trọng, đặc biệt với các cầu thủ tấn công.

5. **Performance: assists** (Số lần kiến tạo): Đặc trưng dạng số thể hiện khả năng hỗ trợ ghi bàn. Cùng với bàn thắng, đây là thước đo chính về hiệu quả đóng góp vào khâu tấn công.

6. **GCA: GCA** (Goal-Creating Actions): Đặc trưng dạng số liên tục, biểu diễn số hành động trực tiếp dẫn đến bàn thắng (ví dụ: đường chuyền quyết định, rê bóng vượt qua đối thủ trước bàn thắng). Là chỉ số tổng hợp cho khả năng sáng tạo.

7. **Progression: PrgR** (Progressive Receptions): Đặc trưng dạng số đo lường số lần cầu thủ nhận bóng ở các vị trí có giá trị tiến bộ (forward-moving). Thể hiện khả năng di chuyển và nhận bóng chiến thuật.

8. **Tackles: Tkl** (Số lần tắc bóng thành công): Đặc trưng dạng số biểu diễn khả năng phòng ngự, đặc biệt quan trọng với các cầu thủ phòng ngự như hậu vệ hoặc tiền vệ phòng ngự.

Tập đặc trưng này bao gồm cả thông tin định lượng và định tính, được chọn lọc để bao phủ nhiều khía cạnh trong lối chơi và hiệu suất thi đấu, từ tấn công, hỗ trợ đến phòng ngự. Việc lựa chọn đặc trưng (feature selection) là một bước quan trọng trong quy trình xây dựng mô hình học máy, đặc biệt đối với các bài toán hồi quy trong bối cảnh dữ liệu thể thao. Các đặc trưng được lựa chọn trong mô hình có tính phù hợp cao dựa trên các tiêu chí sau:

- **Liên quan (Relevance)**: Các đặc trưng được chọn đều là những chỉ số thống kê phổ biến và có cơ sở chuyên môn rõ ràng trong lĩnh vực bóng đá. Cụ thể, tuổi tác, vị trí thi đấu, thời gian thi đấu, khả năng ghi bàn và kiến tạo, chỉ số hỗ trợ tấn công (GCA, PrgR), cũng như chỉ số phòng ngự (Tkl) đều phản ánh trực tiếp đến hiệu suất thi đấu và từ đó ảnh hưởng đến giá trị thị trường của cầu thủ.

- **Tính sẵn có (Availability)**: Các đặc trưng này là dữ liệu được thu thập phổ biến và dễ tiếp cận, thường có trong các cơ sở dữ liệu bóng đá lớn như FBref, Opta hay StatsBomb. Điều này đảm bảo tính khả thi khi triển khai mô hình trong thực tế.
- **Đa dạng (Diversity)**: Bộ đặc trưng bao gồm thông tin đa chiều: thông tin cá nhân (tuổi, vị trí), thời gian thi đấu (minutes), khả năng tấn công (goals, assists, GCA, PrgR) và khả năng phòng ngự (Tkl). Điều này tạo ra một cái nhìn toàn diện về năng lực cầu thủ trên cả khía cạnh chiến thuật và thống kê.
- **Lưu ý trong thực hành (Practical Note)**: Việc lựa chọn đặc trưng không hoàn toàn mang tính cố định mà thường trải qua quá trình thử nghiệm và hiệu chỉnh. Ban đầu có thể dựa trên kiến thức chuyên môn, phân tích dữ liệu khám phá (exploratory data analysis), hoặc sử dụng các kỹ thuật tự động như *Recursive Feature Elimination* hay *feature importance* từ mô hình cây. Việc đánh giá hiệu quả của tập đặc trưng được thực hiện thông qua các chỉ số đánh giá mô hình như MAE, RMSE hoặc R^2.

4.4.2 Code thực tế và mô tả

```
def estimate_player_value(file_path):
    # Load data
    df = pd.read_csv(file_path)

    # Clean minutes column if necessary
    if df['Playing Time: minutes'].dtype == object:
        df['Playing Time: minutes'] = df['Playing Time: minutes'].str.replace(',', '').
            astype(int)

    # Fix Transfer values format to int
    df = df.dropna(subset=['Transfer values'])
    df['Transfer values'] = df['Transfer values'].str.replace('    ', '', regex=False).str
        .replace('M', '', regex=False).astype(float) * 1_000_000
    df['Transfer values'] = df['Transfer values'].astype(int)

    # Select features
    features = [
        'Age',
        'Position',
        'Playing Time: minutes',
        'Performance: goals',
        'Performance: assists',
        'GCA: GCA',
        'Progression: PrgR',
        'Tackles: Tkl'
    ]

    X = df[features]
    y = df['Transfer values']

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

    # Preprocessing: OneHot for categorical 'Position'
    categorical_features = ['Position']
    numeric_features = [col for col in features if col not in categorical_features]

    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ],
        remainder='passthrough'  # Numeric features stay unchanged
    )

    # Model
    model = RandomForestRegressor(n_estimators=100, random_state=42)

```

```
    # Pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', model)
    ])

    # Train
    pipeline.fit(X_train, y_train)

    # Predict and evaluate
    y_pred = pipeline.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    print(f"Mean Absolute Error: {mae:,.0f}    ")

    return pipeline
```

Hàm này thực hiện một quy trình xây dựng và đánh giá một mô hình học máy (Machine Learning) để ước tính giá trị chuyển nhượng (*Transfer values*) của cầu thủ bóng đá dựa trên các số liệu thống kê của họ.

Quy trình hoạt động

1. Tải dữ liệu (Load Data):

```
df = pd.read_csv(file_path)
```

Hàm này đọc dữ liệu từ một file CSV (được chỉ định bởi `file_path`) vào một cấu trúc dữ liệu dạng bảng gọi là DataFrame của thư viện pandas, đặt tên là `df`.

2. Làm sạch dữ liệu (Clean Data): **Xử lý cột 'Playing Time: minutes'**: Kiểm tra kiểu dữ liệu của cột `'Playing Time: minutes'`, nếu kiểu là `'object'` (chuỗi), loại bỏ dấu phẩy và chuyển đổi kiểu dữ liệu thành số nguyên:

```
df['Playing Time: minutes'] = df['Playing Time: minutes'].str.replace(',', '').asty
```

Xử lý cột 'Transfer values': Loại bỏ các giá trị thiếu (`NaN`) và chuyển đổi giá trị chuyển nhượng từ chuỗi (ví dụ: "50.5M") sang số thực:

```
df = df.dropna(subset=['Transfer values'])
df['Transfer values'] = df['Transfer values'].str.replace('€', '', regex=False).str
df['Transfer values'] = df['Transfer values'].astype(int)
```

3. Chọn Features (Đặc trưng - Biến đầu vào): Chọn các đặc trưng có ảnh hưởng đến giá trị chuyển nhượng của cầu thủ:

```
features = ['Age', 'Position', 'Playing Time: minutes', 'Performance: goals', 'Perf
X = df[features]
y = df['Transfer values']
```

4. Phân chia dữ liệu (Split Data): Chia dữ liệu thành tập huấn luyện và tập kiểm tra với tỉ lệ 80/20:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

5. Tiền xử lý dữ liệu (Preprocessing): Xử lý các đặc trưng phân loại và số:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

categorical_features = ['Position']
numeric_features = ['Age', 'Playing Time: minutes', 'Performance: goals', 'Performa
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
        ('num', 'passthrough', numeric_features)
    ])
```

6. Chọn và Khởi tạo Model: Sử dụng mô hình `RandomForestRegressor` với 100 cây quyết định:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
```

7. Tạo Pipeline: Kết hợp các bước tiền xử lý và mô hình vào một Pipeline:

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', model)
])
```

8. Huấn luyện mô hình (Train): Huấn luyện mô hình với dữ liệu huấn luyện:

```
pipeline.fit(X_train, y_train)
```

9. Dự đoán và Đánh giá (Predict and Evaluate): Dự đoán giá trị chuyển nhượng và đánh giá mô hình bằng sai số tuyệt đối trung bình (MAE):

```
from sklearn.metrics import mean_absolute_error
y_pred = pipeline.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error: {mae:,.0f} €")
```

10. Trả về kết quả: Trả về mô hình đã huấn luyện:

```
return pipeline
```

4.4.3 Kiểm thử và đánh giá

Kiểm thử

```
def Task_2():
    # Example usage
    model = estimate_player_value('MoreThan900mins.csv')

    new_player = pd.DataFrame({
        'Age': [26],
        'Position': ['GK'],
        'Playing Time: minutes': [2250],
        'Performance: goals': [0],
        'Performance: assists': [0],
        'GCA: GCA': [0],
        'Progression: PrgR': [0],
        'Tackles: Tkl': [0]
    })

    # D     on    gi    t r    c u    t h    m i
    predicted_value = model.predict(new_player)
    print(f"Estimated player value: {predicted_value[0]:,.0f}    ")
```

Hình 4.3: Terminal sau khi thực hiện function Task_2

chỉ số dòng và sử dụng mã hóa `'utf-8-sig'` để đảm bảo khả năng đọc file tốt hơn (dòng 3).

Kết quả cho thấy thời gian hoàn thành là khoảng 1.92 giây và lượng bộ nhớ tiêu thụ đạt đỉnh ở mức 0.78 MB, cùng với các kết quả tính toán cụ thể như Mean Absolute Error và Estimated player value.

Từ góc độ học thuật, môi trường thực thi được trình bày thông qua giao diện dòng lệnh tiêu chuẩn với tông màu tối. Phân tích kết quả cho thấy script Python đạt hiệu năng thời gian xấp xỉ 1.92 giây và hiệu quả sử dụng bộ nhớ đáng chú ý khi chỉ yêu cầu tối đa 0.78 MB, cung cấp dữ liệu thực nghiệm quan trọng để đánh giá và tối ưu hóa hiệu suất tài nguyên của chương trình.

Đánh giá

Phân tích mã nguồn `Problem4.py` và kết quả thực thi cho thấy chương trình tập trung vào việc xây dựng một mô hình học máy dự đoán giá trị cầu thủ dựa trên các đặc trưng thống kê. Các phần hoạt động ổn định và hiệu quả bao gồm quy trình xử lý dữ liệu ban đầu (tải, làm sạch cơ bản các cột số), lựa chọn đặc trưng, chia tách tập dữ liệu, và đặc biệt là việc triển khai pipeline học máy sử dụng `scikit-learn`. Pipeline này kết hợp tiền xử lý dữ liệu phân loại bằng One-Hot Encoding thông qua `ColumnTransformer` và mô hình hồi quy `Random Forest`.

Kết quả thực thi từ terminal xác nhận tính năng động của pipeline này, hiển thị Lỗi tuyệt đối trung bình (Mean Absolute Error - MAE) là **15.512.382 €**, một chỉ số định lượng về độ chính xác của mô hình trên tập dữ liệu kiểm tra. Giá trị cầu thủ ước tính cho một trường hợp cụ thể là **20.391.000 €**. Chương trình cũng tích hợp chức năng đo lường hiệu năng, ghi nhận thời gian chạy là khoảng **1.92 giây** và mức sử dụng bộ nhớ đạt đỉnh là **0.78 MB**, cho thấy hiệu quả tương đối về tài nguyên đối với quy mô dữ liệu được xử lý trong lần chạy này.

Ưu điểm của mã nguồn:

- **Tính mô-đun**: Mã được tổ chức thành các hàm riêng biệt cho từng tác vụ (lấy dữ liệu, cập nhật dữ liệu, lưu kết quả, xây dựng mô hình, các task cụ thể), giúp tăng khả năng đọc và bảo trì.
- **Sử dụng thư viện chuẩn mạnh mẽ**: Tận dụng hiệu quả các thư viện phổ biến và tối ưu hóa cao như `pandas` cho xử lý dữ liệu, `scikit-learn` cho học máy (bao gồm `Pipeline`, `ColumnTransformer`, `RandomForestRegressor`), và `time`/`tracemalloc` cho phân tích hiệu năng.
- **Triển khai Pipeline học máy**: Việc sử dụng Pipeline và ColumnTransformer là phương pháp tốt để xử lý các bước tiền xử lý và mô hình hóa một cách có hệ thống, giúp tránh rò rỉ dữ liệu (data leakage) giữa các bước, đặc biệt quan trọng trong quá trình đánh giá mô hình.

- **Đo lường hiệu năng tích hợp**: Việc thêm các đoạn mã đo thời gian chạy và bộ nhớ sử dụng là một điểm cộng lớn, cung cấp thông tin cần thiết để đánh giá và tối ưu hóa hiệu quả tài nguyên của chương trình.

Nhược điểm của mã nguồn:

- **Độ chính xác**: Chỉ số MAE là **15.512.382 €** là một con số cụ thể, nhưng việc đánh giá "độ chính xác cao"hay thấp phụ thuộc vào bối cảnh của bài toán (phạm vi giá trị chuyển nhượng, tính biến động của dữ liệu, so sánh với các mô hình khác hoặc baseline). Nếu không có thêm thông tin so sánh, khó có thể kết luận về tính hiệu quả thực sự của mô hình.

- **Phụ thuộc vào nguồn dữ liệu ngoại vi**: Hàm `update_data` phụ thuộc mạnh mẽ vào cấu trúc HTML của website `footballtransfers.com`. Bất kỳ thay đổi nào trên website đều có thể làm hỏng chức năng crawl dữ liệu, khiến phần này trở nên kém bền vững.

- **Thiếu xử lý lỗi mạnh mẽ**: Mặc dù có xử lý `try...except` cơ bản trong phần crawl, mã chưa có cơ chế xử lý lỗi toàn diện hơn cho các trường hợp như tệp dữ liệu đầu vào không tồn tại, định dạng dữ liệu không đúng, hoặc lỗi kết nối khi crawl.

- **Thiếu tùy chỉnh và đánh giá mô hình sâu hơn**: Mã chỉ sử dụng một loại mô hình (`RandomForestRegressor`) với các tham số mặc định (`n_estimators=100`) và chỉ đánh giá trên một lần chia tách dữ liệu (`test_size=0.2`). Việc khám phá các mô hình khác, điều chỉnh siêu tham số (hyperparameter tuning), và sử dụng Cross-validation sẽ cung cấp đánh giá độ tin cậy và tiềm năng cải thiện hiệu suất mô hình tốt hơn.

- **Đường dẫn tệp cố định**: Các đường dẫn tệp đầu vào và đầu ra được mã hóa cứng, làm giảm tính linh hoạt và khả năng tái sử dụng của mã trên các hệ thống hoặc cấu trúc thư mục khác nhau.

Tóm lại, mã nguồn thể hiện sự hiểu biết về quy trình xử lý dữ liệu và xây dựng mô hình học máy bằng các thư viện chuẩn. Các phần cốt lõi liên quan đến ML pipeline và đo lường hiệu năng hoạt động tốt. Tuy nhiên, tính phụ thuộc vào nguồn dữ liệu ngoại vi không ổn định, sự thiếu vắng xử lý lỗi toàn diện và việc đánh giá mô hình chưa chuyên sâu là những điểm có thể cải thiện.

Kết luận

Báo cáo này đã trình bày quá trình thực hiện một loạt các nhiệm vụ phân tích dữ liệu bóng đá sử dụng ngôn ngữ lập trình Python, tập trung vào việc thu thập, xử lý, phân tích và mô hình hóa dữ liệu thống kê cầu thủ từ giải Ngoại hạng Anh mùa giải 2004–2005 và dữ liệu chuyển nhượng gần đây.

Tóm tắt kết quả chính:

Thu thập và Tiền xử lý Dữ liệu (Vấn đề I): Đã thành công trong việc thu thập dữ liệu thống kê chi tiết cho 491 cầu thủ thi đấu trên 90 phút từ trang `fbref.com` bằng kỹ thuật web scraping với Selenium và BeautifulSoup. Dữ liệu được làm sạch, chuẩn hóa (các giá trị không có sẵn được đánh dấu "N/a") và lưu trữ có cấu trúc trong tệp `results.csv`. Quá trình này đảm bảo tính chính xác và đầy đủ của dữ liệu nhưng cho thấy hiệu suất thời gian chưa tối ưu (~186 giây) do việc khởi tạo nhiều trình duyệt. Tuy nhiên, việc sử dụng bộ nhớ lại khá hiệu quả (~32.74 MB).

Phân tích Thống kê Mô tả (Vấn đề II): Đã tiến hành phân tích sâu hơn trên tệp `results.csv`. Xác định được top 3 cầu thủ có chỉ số cao nhất và thấp nhất cho từng thống kê (lưu tại `top_3.txt`). Tính toán các đại lượng thống kê mô tả quan trọng (trung vị, trung bình, độ lệch chuẩn) cho toàn giải đấu và từng đội (lưu tại `results2.csv`). Trực quan hóa sự phân phối của các chỉ số chính thông qua biểu đồ histogram (lưu dưới dạng file PNG). Dựa trên số lượng chỉ số dẫn đầu, Liverpool được xác định là đội có hiệu suất tổng thể tốt nhất (dẫn đầu 28 chỉ số, kết quả lưu tại `best_team_summary.txt`). Phân tích này cung cấp cái nhìn đa chiều về hiệu suất cầu thủ và đội bóng, dù phương pháp xác định đội tốt nhất còn tương đối đơn giản.

Phân cụm Cầu thủ (Vấn đề III): Áp dụng thuật toán K-means để phân nhóm cầu thủ dựa trên dữ liệu thống kê. Số cụm tối ưu được xác định là $K = 6$ thông qua phương pháp Elbow và Silhouette Score, phù hợp với các nhóm vị trí thực tế trong bóng đá. Kỹ thuật PCA được sử dụng để giảm chiều dữ liệu xuống 2 chiều, cho phép trực quan hóa các cụm cầu thủ một cách hiệu quả. Phân tích đặc điểm của từng cụm cho thấy sự tương đồng về vai trò hoặc phong cách chơi của các cầu thủ trong cùng một nhóm (kết quả phân tích và biểu đồ lưu trong thư mục `P3_RES`). Mặc dù K-means và PCA có những hạn chế nhất định (giả định về hình dạng cụm, mất mát thông tin), phương pháp này đã cung cấp những hiểu biết ban đầu hữu ích về cấu trúc tiềm ẩn trong dữ liệu cầu thủ.

Ước tính Giá trị Cầu thủ (Vấn đề IV): Đã thu thập dữ liệu giá trị chuyển nhượng từ `footballtransfers.com` cho các cầu thủ thi đấu trên 900 phút (lưu tại `MoreThan900mins.csv`). Đề xuất và triển khai thành công mô hình `RandomForestRegressor` để ước tính giá trị cầu thủ. Các đặc trưng đầu vào quan trọng như Tuổi, Vị trí, Thời gian thi đấu, Bàn thắng, Kiến tạo, GCA, PrgR, Tkl đã được lựa chọn dựa trên kiến thức chuyên môn và nghiên cứu. Mô hình đạt được Sai số Tuyệt đối Trung bình (MAE) khoảng €15.5 triệu trên tập kiểm tra. Việc sử dụng `Pipeline` trong Scikit-learn giúp quy trình tiền xử lý và

huấn luyện được thực hiện một cách có hệ thống. Tuy nhiên, mô hình còn phụ thuộc vào nguồn dữ liệu bên ngoài và cần cải thiện về độ chính xác cũng như đánh giá sâu hơn (ví dụ: tinh chỉnh siêu tham số, sử dụng cross-validation).

Đánh giá chung:

Tổng thể, dự án đã hoàn thành tốt các mục tiêu đề ra, thể hiện được năng lực áp dụng các kỹ thuật khoa học dữ liệu từ thu thập, xử lý, phân tích đến mô hình hóa trong lĩnh vực thể thao. Việc sử dụng hiệu quả các thư viện Python như `Pandas`, `Scikit-learn`, `Selenium`, `BeautifulSoup` là một điểm mạnh nổi bật. Các kết quả phân tích và mô hình hóa cung cấp những thông tin giá trị về hiệu suất cầu thủ, đặc điểm đội bóng và các yếu tố ảnh hưởng đến giá trị chuyển nhượng.

Tuy nhiên, vẫn còn một số điểm có thể cải thiện như tối ưu hóa tốc độ web scraping, xây dựng các chỉ số đánh giá tổng hợp phức tạp hơn, và thực hiện đánh giá mô hình một cách toàn diện hơn. Sự phụ thuộc vào cấu trúc các trang web bên ngoài cũng là một yếu điểm cần lưu ý về tính bền vững của giải pháp thu thập dữ liệu.

Nhìn chung, báo cáo này là một minh chứng cho khả năng ứng dụng mạnh mẽ của Python và các kỹ thuật học máy trong việc khám phá và rút ra tri thức từ dữ liệu bóng đá phức tạp.

Danh mục tài liệu tham khảo

1. Scikit-learn Developers. (n.d.). RandomForestRegressor. Scikit-learn documentation.

2. GeeksforGeeks. (n.d.). Random Forest Regression in Python.

3. Université du Luxembourg. (2020). Random Forest Regression [Project Report].

4. StatQuest with Josh Starmer. (n.d.). Random Forests Part 1: Regression [Video]. YouTube.

5. Metelski, Adam. (2021). Factors affecting the value of football players in the transfer market. Journal of Physical Education and Sport. 21. 1150-1155. 10.7752/jpes.2021.s2145.

6. Rong, Zhangyi, Wang, Lujie, & Xie, Shengting. (2024). *Factors that Influence Player Market Value in Different Position: Evidence from European Leagues.* Advances in Economics, Management and Political Sciences, 82, 50–63.

7. Football Benchmark. (n.d.). Game changers: A snapshot of the top 100 most valuable players. Football Benchmark.

8. Zhang, Y., & Zhang, W. (2019). *Research on player value evaluation model based on multiple linear regression analysis.*

9. Krabbenborg, L. (2020). The relationship between player market value and performance in professional European football: A statistical analysis (Bachelor's thesis, Tilburg University).