

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CNTT

-----□□□□-----



BÁO CÁO BÀI TẬP LỚN

Môn học: Lập trình python

Giảng viên hướng dẫn: Kim Xuân Bách

Thành viên:

B23DCCN293 - Mạc Đăng Hiệp

B23DCCN520 - Nguyễn Thế Lưu

B23DCCN919 - Nguyễn Hữu Việt

Câu I.1 Thu thập dữ liệu thống kê của các cầu thủ

1. Lý giải cách làm

Để thực hiện, nhóm lựa chọn cách làm như sau:

- Sử dụng Selenium kết hợp webdriver-manager để điều khiển trình duyệt Chrome tự động. Fbref tải bảng thống kê bằng JavaScript nên nếu chỉ dùng requests có thể không lấy đủ dữ liệu. Webdriver-manager giúp tự động tải về phiên bản ChromeDriver phù hợp với trình duyệt đang cài trên máy.
- Với mỗi URL thống kê trong từ điển STATS_URLS (standard, shooting, passing, gca, defense, possession, misc, keeper), chương trình gọi hàm get_html(url) để mở trang bằng Selenium, chờ phần tử bảng table.stats_table xuất hiện (WebDriverWait), sau đó đợi thêm vài giây rồi lấy toàn bộ HTML bằng driver.page_source.
- HTML thu được được phân tích bằng thư viện BeautifulSoup. Chương trình tìm bảng thống kê theo id stats_<loại_thống_kê>; nếu không tìm được thì fallback tìm bảng có class stats_table. Từ bảng này, chương trình lấy phần tbody và duyệt từng hàng tr (mỗi hàng tương ứng một cầu thủ).
- Dữ liệu được tổ chức theo cấu trúc từ điển players_data, trong đó key là tên cầu thủ, value là một từ điển con chứa các chỉ số. Để ánh xạ giữa tên thuộc tính data-stat trên trang web và tên cột trong dữ liệu, em dùng một từ điển COLUMN_MAP. Mỗi mục trong COLUMN_MAP có dạng (tên_cột_trong_kết_quả, kiểu_dữ_liệu). Khi gặp một cầu thủ mới, chương trình khởi tạo tất cả các cột trong bản ghi của cầu thủ đó bằng 'N/A' rồi gán cột Player bằng tên cầu thủ. Sau đó, mỗi lần đọc được một ô thống kê tương ứng với data-stat nằm trong COLUMN_MAP, giá trị sẽ được ghi đè lên 'N/A' cho cột đó.
- Hàm parse_value có nhiệm vụ làm sạch và ép kiểu giá trị thô đọc từ HTML. Hàm loại bỏ các ký tự như dấu phẩy, dấu phần trăm, dấu cộng; kiểm tra chuỗi rỗng; ép kiểu sang int hoặc float nếu cần, hoặc trả về N/A nếu không ép kiểu được. Với các giá trị phần trăm (ví dụ 50%), hàm chuyển về dạng số thực phù hợp. Một số cột đặc biệt được xử lý riêng, ví dụ cột tuổi age thường có dạng '24-123' nên chỉ giữ phần trước dấu gạch ngang; cột nationality dùng regex để chỉ giữ lại mã quốc gia 3 chữ cái.
- Sau khi lần lượt duyệt qua tất cả các loại bảng thống kê, bản ghi của mỗi cầu thủ trong players_data sẽ được ghép từ nhiều nguồn khác nhau: bảng standard cung cấp phút thi đấu, số bàn, kiến tạo...; bảng shooting cung cấp các chỉ số sút; bảng passing cung cấp các chỉ số chuyền bóng; bảng defense, possession,

misc, keeper cung cấp các chỉ số phòng ngự, kiểm soát bóng, thống kê khác và thủ môn. Những chỉ số không tồn tại trên trang hoặc không áp dụng cho cầu thủ đó sẽ giữ nguyên giá trị 'N/A'.

- Ở bước cuối, chương trình duyệt toàn bộ `players_data`, chỉ chọn các cầu thủ có số phút thi đấu `Standard_Min` lớn hơn 90 để đưa vào danh sách `filtered_data`. Danh sách này được chuyển thành pandas DataFrame với các cột lấy từ `COLUMN_MAP`, sau đó điền giá trị N/A cho các ô còn thiếu, sắp xếp theo tên cầu thủ và lưu ra file CSV `results.csv`.

2. Thuật toán áp dụng

Thuật toán tổng quát của chương trình có thể tóm tắt theo các bước sau:

Bước 1: Khởi tạo WebDriver

- Gọi `ChromeDriverManager().install()` để tải và cấu hình ChromeDriver.
- Tạo đối tượng `driver = webdriver.Chrome(service=service)`. Nếu có lỗi (không cài Chrome, sai phiên bản), chương trình in ra thông báo và kết thúc.

Bước 2: Khởi tạo cấu trúc dữ liệu

- Khai báo `STATS_URLS` chứa các đường link thống kê cho từng loại chỉ số.
- Khai báo `COLUMN_MAP` ánh xạ giữa data-stat trên HTML và (tên cột, kiểu dữ liệu) trong bảng kết quả.
- Khởi tạo `players_data = {}` để lưu dữ liệu của tất cả cầu thủ, `debug_info` để ghi lại thông tin phục vụ việc kiểm tra.

Bước 3: Thu thập dữ liệu từ từng loại thống kê

Đối với mỗi cặp (`stat_type`, `url`) trong `STATS_URLS`, thực hiện:

- Gọi hàm `get_html(url)`: điều khiển Selenium mở trang, chờ xuất hiện bảng `table.stats_table`, đợi thêm một khoảng thời gian rồi lấy mã HTML.

- Dùng BeautifulSoup để phân tích HTML, tìm bảng thống kê theo id `stats_stat_type`; nếu không thấy thì tìm bảng có class `stats_table`. Lấy phần `tbody` của bảng.

- Duyệt từng hàng tr trong `tbody`. Với mỗi hàng:

- Tìm ô `player_cell` với `data-stat='player'` để lấy tên cầu thủ. Nếu không có, bỏ qua hàng.

- Nếu cầu thủ chưa có trong `players_data`, khởi tạo một bản ghi mới với tất cả cột bằng 'N/A' và gán cột Player là tên cầu thủ.

- Duyệt tất cả các ô (`th`, `td`) trong hàng: lấy thuộc tính `stat = cell.get('data-stat')`. Nếu `stat` xuất hiện trong `COLUMN_MAP` thì:

- + Lấy tên cột và kiểu dữ liệu tương ứng.
- + Lấy nội dung thô `raw_value = cell.text.strip()`.
- + Xử lý đặc biệt cho một số cột như `age`, `nationality` nếu cần.
- + Gọi `parse_value(raw_value, dtype)` để làm sạch và ép kiểu.
- + Gán giá trị đã xử lý vào `players_data[player_name][Tên_cột]`.

Bước 4: Lọc cầu thủ theo số phút thi đấu

- Sau khi đã duyệt xong tất cả các loại bảng thống kê, tiến hành lọc dữ liệu:
 - Khởi tạo `filtered_data = []`, `players_with_insufficient_data = 0`.
 - Với mỗi cầu thủ trong `players_data`, lấy `minutes_played = player_data.get('Standard_Min', 0)`.
 - Nếu `minutes_played` khác 'N/A' và lớn hơn 90, tạo một danh sách `ordered_row` chứa các giá trị của cầu thủ đó theo đúng thứ tự cột trong `COLUMN_MAP` rồi thêm vào `filtered_data`. Ngược lại, tăng biến đếm `players_with_insufficient_data`.

Bước 5: Tạo bảng kết quả và lưu dữ liệu

- Tạo danh sách tên cột từ `COLUMN_MAP` và khởi tạo pandas DataFrame từ `filtered_data`.
- Gọi `df.fillna('N/A')` để đảm bảo các giá trị thiếu đều hiển thị là N/A.
- Sắp xếp DataFrame theo cột `Player` để bảng dễ quan sát.
- Lưu DataFrame ra file CSV `results.csv` trong cùng thư mục với file chương trình.
- In ra thông tin tổng kết số cầu thủ được lưu và số cầu thủ bị loại do chơi ít hơn hoặc bằng 90 phút.

Bước 6: Kết thúc chương trình

- Đóng trình duyệt tự động bằng `driver.quit()` để giải phóng tài nguyên.

I.2

1. Lý giải cách làm

Ở phần I.2, yêu cầu là thu thập giá chuyển nhượng của các cầu thủ Premier League mùa 2024–2025 từ trang [footballtransfers.com](https://www.footballtransfers.com), dựa trên danh sách cầu thủ đã có trong file `results.csv` (kết quả của phần I.1), sau đó lưu kết quả vào file `transfer_values.csv`. Trong quá trình crawl có thể gặp CAPTCHA hoặc rate-limit

nên chương trình cần có cơ chế xử lý và đề xuất biện pháp khắc phục.

Cách làm tổng quát như sau:

- Chương trình đọc danh sách cầu thủ từ file results.csv bằng pandas. Mỗi dòng trong DataFrame tương ứng với một cầu thủ, với các cột quan trọng là Player (tên cầu thủ) và Team (tên câu lạc bộ).
- Em sử dụng Selenium kết hợp webdriver-manager để điều khiển trình duyệt Chrome. Trong hàm setup_driver, chương trình cấu hình ChromeOptions với các tham số chống bị phát hiện tự động như tắt AutomationControlled, tắt automation extension, chỉnh kích thước cửa sổ và random User-Agent. Sau khi tạo driver, chương trình còn ghi đè thuộc tính navigator.webdriver bằng JavaScript để hạn chế việc website nhận diện trình duyệt tự động.
- Với mỗi cầu thủ, chương trình ưu tiên lấy dữ liệu từ footballtransfers.com thông qua hàm search_player_value. Hàm này đầu tiên thử truy cập trực tiếp trang cầu thủ bằng cách dựng URL dạng /en/players/firstname-lastname. Nếu trang tồn tại, chương trình dùng BeautifulSoup phân tích HTML và gọi hàm extract_value_from_page để trích ra giá trị chuyển nhượng. Nếu không thành công, hàm chuyển sang phương án tìm kiếm: truy cập trang search của footballtransfers với từ khóa là tên cầu thủ, lấy kết quả đầu tiên, truy cập vào trang cầu thủ tương ứng và tiếp tục trích xuất giá trị.
- Hàm extract_value_from_page cố gắng tìm các thành phần HTML thể hiện giá trị chuyển nhượng bằng nhiều cách: tìm các div/span có class gợi ý market value, tìm các chuỗi văn bản chứa ký hiệu tiền tệ (€, \$, £) kèm đơn vị M hoặc K, hoặc tìm trong các thuộc tính data-value. Nếu bắt được một chuỗi phù hợp, hàm dùng biểu thức chính quy để cắt ra phần giá trị, ví dụ "€45M".
- Để phát hiện CAPTCHA, chương trình sử dụng hàm check_for_captcha. Hàm này kiểm tra page source và tiêu đề trang để tìm các cụm từ như "captcha", "recaptcha", "cloudflare", "just a moment", "verify you are human"... Nếu phát hiện CAPTCHA, hàm search_player_value trả về mã "CAPTCHA" để thông báo cho hàm main biết.
- Khi gặp CAPTCHA, thay vì dừng toàn bộ chương trình, mã nguồn chuyển sang sử dụng nguồn dự phòng là Transfermarkt thông qua hàm get_transfermarkt_value. Hàm này gửi yêu cầu HTTP tới trang tìm kiếm nhanh của Transfermarkt bằng thư viện requests, sau đó phân tích HTML bằng BeautifulSoup và cố gắng lấy ô market value đầu tiên (thường là thẻ td có class 'rechts hauptlink'). Giá trị lấy được sẽ dùng làm giá chuyển nhượng cho cầu thủ tương ứng. Đây là cách giảm số lần truy cập footballtransfers nhưng vẫn có dữ

liệu cho nhiều cầu thủ.

- Sau mỗi lần lấy giá (hoặc N/A nếu không tìm thấy), chương trình thêm một bản ghi vào danh sách `transfer_data` gồm các trường `Player`, `Team` và `Transfer_Value_2024_25`. Để hạn chế bị chặn do gửi quá nhiều yêu cầu, em chèn các khoảng nghỉ ngẫu nhiên 1–3 giây giữa các cầu thủ và nghỉ lâu hơn 15–30 giây sau mỗi 15 cầu thủ. Đồng thời, chương trình có cơ chế checkpoint: cứ sau mỗi 10 cầu thủ sẽ lưu tạm dữ liệu vào file `transfer_checkpoint.csv`; nếu chương trình bị dừng, lần chạy sau có thể đọc checkpoint và tiếp tục từ vị trí đã xử lý.

- Cuối cùng, toàn bộ danh sách `transfer_data` được chuyển thành `DataFrame` và lưu ra file `transfer_values.csv`. Chương trình in ra thống kê tổng số cầu thủ, số cầu thủ có giá trị khác N/A và số cầu thủ không tìm thấy giá. Driver Selenium được đóng lại bằng `driver.quit()` để giải phóng tài nguyên.

2. Thuật toán áp dụng

Thuật toán của chương trình phần I.2 có thể mô tả theo các bước sau:

Bước 1: Chuẩn bị dữ liệu đầu vào

- Xác định đường dẫn tới `results.csv`, `transfer_values.csv` và `transfer_checkpoint.csv` trong cùng thư mục với file code.
- Kiểm tra sự tồn tại của `results.csv`. Nếu không có, in thông báo lỗi và kết thúc chương trình.
- Đọc `results.csv` bằng `pandas.read_csv` để lấy danh sách cầu thủ. Mỗi hàng có ít nhất cột `Player` và `Team`.
- Nếu tồn tại file checkpoint, hỏi người dùng có muốn tiếp tục (resume) hay không. Nếu có, đọc checkpoint vào `DataFrame`, chuyển thành list dict và đặt `start_idx` bằng số bản ghi đã xử lý.

Bước 2: Khởi tạo WebDriver

- Gọi `setup_driver` để khởi tạo `ChromeDriver` với các tùy chọn chống bị phát hiện: tắt `AutomationControlled`, tắt extension automation, random User-Agent, override `navigator.webdriver`.

Bước 3: Vòng lặp xử lý từng cầu thủ

- Duyệt chỉ số `idx` từ `start_idx` đến hết `DataFrame` đầu vào:
 - Lấy `player_name = row['Player']` và `team_name = row.get('Team', 'Unknown')`.

- Gọi `search_player_value(driver, player_name, team_name, use_backup=False)` để cố gắng lấy giá từ `footballtransfers`.
- Nếu kết quả là 'CAPTCHA', in cảnh báo, sau đó gọi `get_transfermarkt_value(player_name, team_name)` để lấy giá từ `Transfermarkt` cho riêng cầu thủ này. Sau khi lấy giá, chờ thêm 10–20 giây trước khi quay lại `footballtransfers` cho cầu thủ tiếp theo.
- Nếu kết quả không phải 'CAPTCHA', đó sẽ là giá trị dạng chuỗi (ví dụ "€45M") hoặc 'N/A' nếu không tìm thấy.
- Thêm dict `{'Player': player_name, 'Team': team_name, 'Transfer_Value_2024_25': transfer_value}` vào `list transfer_data`.
- Sau mỗi 10 cầu thủ, tạo `DataFrame` từ `transfer_data` và ghi ra `transfer_checkpoint.csv` để lưu tiến độ.
- Chèn `delay` ngẫu nhiên 1–3 giây giữa các cầu thủ, và `delay` dài 15–30 giây sau mỗi 15 cầu thủ để giảm nguy cơ bị chặn.

Bước 4: Hàm `search_player_value`

- Nếu `use_backup=True` thì chỉ gọi `get_transfermarkt_value` và trả về kết quả.
- Ngược lại, hàm thực hiện:
 - Chuẩn hóa tên cầu thủ, tách thành các từ, dựng URL trực tiếp dạng `/en/players/firstname-lastname` và truy cập bằng `driver.get`.
 - Đợi một khoảng thời gian, kiểm tra tiêu đề và nội dung để xem có phải trang 404 không; nếu hợp lệ, dùng `BeautifulSoup` và `extract_value_from_page` để trích giá. Nếu trích được thì trả về.
 - Nếu không thành công, dựng URL tìm kiếm `https://www.footballtransfers.com/en/search?q=<tên_cầu_thủ>` và truy cập.
 - Ngay sau khi tải trang `search`, gọi `check_for_captcha`; nếu phát hiện CAPTCHA thì trả về 'CAPTCHA'.
 - Nếu không có CAPTCHA, dùng `BeautifulSoup` tìm các link cầu thủ trong kết quả, chọn link đầu tiên, truy cập trang cầu thủ và một lần nữa kiểm tra CAPTCHA. Nếu vẫn không bị CAPTCHA, gọi `extract_value_from_page` để trích giá. Nếu mọi phương án đều thất bại, trả về 'N/A'.

Bước 5: Hàm `get_transfermarkt_value` (nguồn dự phòng)

- Dựng URL tới trang tìm kiếm nhanh của `Transfermarkt` với query là tên cầu thủ.
- Gửi HTTP request bằng `requests` với header `User-Agent` và `Accept-Language`.
- Nếu phản hồi thành công (status code 200), phân tích HTML bằng

BeautifulSoup, tìm ô market value đầu tiên (thường là thẻ td class='rechts hauptlink') và lấy text làm giá trị. Nếu không tìm thấy hoặc gặp lỗi, trả về 'N/A'.

Bước 6: Ghi kết quả và kết thúc

- Sau khi xử lý hết cầu thủ, chuyển transfer_data thành DataFrame và ghi ra transfer_values.csv.
- Nếu tồn tại file checkpoint, xóa file này vì tiến trình đã hoàn tất.
- In ra thông kê tổng số cầu thủ, số cầu thủ có giá trị khác 'N/A' và số cầu thủ không tìm được giá.
- Trong khối finally, gọi driver.quit() để đóng trình duyệt và giải phóng tài nguyên.

Kết quả chạy được:

```
Crawling: passing...
  Processed data from passing table
Crawling: gca...
  Processed data from gca table
Crawling: defense...
  Processed data from defense table
Crawling: possession...
  Processed data from possession table
Crawling: misc...
  Processed data from misc table
Crawling: keeper...
  Processed data from keeper table

=== SUMMARY ===
Data for 494 players saved to d:\VIEnguyen72 Downloads\BTL-Python-main\BTL-Python-main\main\results.csv
Players excluded due to insufficient minutes (< 90): 69
PS D:\VIEnguyen72 Downloads\BTL-Python-main> █
```


trả dữ liệu cho client, mỗi cầu thủ sẽ có thêm cột giá trị chuyển nhượng nếu đã được crawl ở phần I.2.

Ứng dụng Flask định nghĩa các endpoint chính:

- '/' (index): trả về thông tin giới thiệu API, liệt kê các endpoint, tổng số cầu thủ và số câu lạc bộ.
- '/api/player/<player_name>': tra cứu toàn bộ chỉ số của cầu thủ theo tên.
- '/api/club/<club_name>': tra cứu toàn bộ chỉ số của các cầu thủ thuộc một câu lạc bộ.
- '/api/stats': trả về một số thống kê tổng quan (tổng số cầu thủ, câu lạc bộ, tổng bàn thắng, tổng kiến tạo, cầu thủ ghi nhiều bàn nhất, cầu thủ kiến tạo nhiều nhất...).

Ở endpoint `/api/player/<player_name>`, chương trình tìm cầu thủ theo tên không phân biệt hoa thường bằng cách so sánh cột `Player` sau khi chuyển về lowercase với tham số `player_name`. Nếu không tìm thấy, code thử lại bằng `str.contains` để cho phép khớp một phần tên (partial match). Khi có dữ liệu, chương trình gọi `merge_player_data` để gộp thêm cột giá chuyển nhượng, sau đó chuyển `DataFrame` sang danh sách dict bằng `to_dict('records')` và thay các giá trị NaN bằng chuỗi 'N/A' trước khi trả về JSON. Nếu không tìm thấy cầu thủ phù hợp, API trả về JSON lỗi 404 kèm thông báo gợi ý.

Endpoint `/api/club/<club_name>` hoạt động tương tự nhưng lọc theo cột `Team`. Chương trình cũng hỗ trợ khớp chính xác (lowercase) và khớp một phần tên đội (contains). Nếu không có đội bóng nào phù hợp, API trả về danh sách tất cả các câu lạc bộ hiện có trong dữ liệu để người dùng tham khảo. Khi có dữ liệu, kết quả được merge với bảng giá trị chuyển nhượng, sắp xếp theo tên cầu thủ rồi trả về JSON chứa tên câu lạc bộ, tổng số cầu thủ và danh sách chi tiết từng cầu thủ.

Endpoint `/api/stats` đọc trực tiếp từ `player_stats_df` để tính các thống kê tổng quan: tổng số cầu thủ, số câu lạc bộ, danh sách tên câu lạc bộ, tổng số bàn thắng và kiến tạo, đồng thời xác định cầu thủ ghi nhiều bàn nhất và cầu thủ kiến tạo nhiều nhất (nếu các cột tương ứng tồn tại). Ứng dụng cũng định nghĩa các handler cho lỗi 404 và 500 để trả về thông báo JSON thân thiện khi người dùng gọi sai URL hoặc xảy ra lỗi nội bộ.

Cuối cùng, trong khối `if __name__ == '__main__':`, chương trình gọi

load_data() để nạp dữ liệu rồi chạy app.run(host='0.0.0.0', port=5000, debug=True). Khi server chạy, người dùng có thể gọi trực tiếp các endpoint qua trình duyệt hoặc dùng chương trình lookup ở phần II.2 để tra cứu và xuất kết quả.

2. Thuật toán áp dụng

Thuật toán chính của phần II.1 có thể mô tả theo các bước sau:

Bước 1: Nạp dữ liệu

- Xác định SCRIPT_DIR và đường dẫn tới results.csv, transfer_values.csv.
- Hàm load_data() kiểm tra sự tồn tại của từng file:
 - + Nếu có results.csv: đọc vào player_stats_df bằng pandas.read_csv; nếu không, tạo DataFrame rỗng.
 - + Nếu có transfer_values.csv: đọc vào transfer_values_df; nếu không, tạo DataFrame rỗng.

Bước 2: Hàm merge_player_data(stats_data)

- Nếu transfer_values_df không rỗng, dùng pandas.merge để nối stats_data với transfer_values_df trên cột Player theo kiểu how='left', chỉ giữ thêm cột Transfer_Value_2024_25.
- Nếu transfer_values_df rỗng, trả lại stats_data không thay đổi.

Bước 3: Endpoint index '/'

- Khi nhận request GET, tạo một dict chứa message giới thiệu, danh sách endpoint, tổng số cầu thủ và số câu lạc bộ (dựa trên player_stats_df), sau đó jsonify và trả về cho client.

Bước 4: Endpoint '/api/player/<player_name>'

- Nếu player_stats_df rỗng: trả JSON lỗi 500.
- Lọc DataFrame theo tên cầu thủ (không phân biệt hoa thường):
 - + player_data = player_stats_df[player_stats_df['Player'].str.lower() == player_name.lower()].
- Nếu player_data rỗng, thử lại bằng partial match:
 - + player_data = player_stats_df[player_stats_df['Player'].str.contains(player_name, case=False, na=False)].

- Nếu vẫn rỗng: trả JSON lỗi 404 với thông báo không tìm thấy cầu thủ.
- Nếu tìm thấy:
 - + Gọi `merge_player_data(player_data)` để gộp thêm cột `Transfer_Value_2024_25`.
 - + Chuyển DataFrame kết quả sang list dict bằng `to_dict('records')`.
 - + Duyệt từng record, nếu giá trị là NaN thì thay bằng chuỗi 'N/A'.
 - + Nếu chỉ có 1 bản ghi, trả JSON `{'success': True, 'player': record}`; nếu nhiều, trả JSON chứa danh sách `'players'`.

Bước 5: Endpoint `'/api/club/<club_name>'`

- Nếu `player_stats_df` rỗng: trả JSON lỗi 500.
- Lọc DataFrame theo cột Team tương tự như với Player (so sánh lowercase, sau đó dùng `contains` nếu cần).
- Nếu sau cả hai bước lọc mà vẫn rỗng: lấy danh sách tất cả Team bằng `player_stats_df['Team'].unique()`, sắp xếp và trả JSON lỗi 404 kèm `available_clubs`.
- Nếu có dữ liệu:
 - + Gọi `merge_player_data(club_data)` để gộp thêm giá trị chuyển nhượng.
 - + Sắp xếp theo cột Player.
 - + Chuyển sang list dict, thay NaN bằng 'N/A' rồi trả JSON gồm: `success=True`, tên câu lạc bộ, `total_players` và danh sách `players`.

Bước 6: Endpoint `'/api/stats'`

- Nếu `player_stats_df` rỗng: trả JSON lỗi 500.
- Nếu không:
 - + Tính `total_players = len(player_stats_df)`.
 - + Tính `total_clubs =` số lượng Team khác nhau.
 - + Lấy danh sách clubs = `sorted(player_stats_df['Team'].unique())`.
 - + Nếu tồn tại cột `Standard_Gls`: tính tổng bàn thắng và tìm cầu thủ có số bàn tối đa (`idxmax`). Nếu không, đặt 'N/A'.
 - + Tương tự cho cột `Standard_Ast` để lấy tổng kiến tạo và cầu thủ kiến tạo nhiều nhất.
- Trả JSON stats cho client.

Bước 7: Xử lý lỗi và chạy server

- Định nghĩa `errorhandler` cho 404: trả JSON thông báo endpoint không tồn tại và liệt kê các endpoint hợp lệ.

- Định nghĩa errorhandler cho 500: trả JSON báo lỗi nội bộ cùng message lỗi.
- Trong khối main, gọi load_data(), in thông tin hướng dẫn rồi chạy app.run(host='0.0.0.0', port=5000, debug=True).

Câu II.2:

1. Lý giải cách làm

Ở phần II.2, yêu cầu là xây dựng một chương trình Python (lookup.py) sử dụng module requests để gọi các API đã xây dựng ở phần II.1, cho phép người dùng tra cứu thông tin cầu thủ hoặc câu lạc bộ từ dòng lệnh. Kết quả cần được hiển thị ra màn hình dưới dạng bảng và đồng thời lưu vào một file CSV tương ứng.

Trong file lookup.py, em hiện thực chương trình với các ý chính như sau:

- Cấu hình API: Đặt hằng số API_BASE_URL = "http://127.0.0.1:5000" trỏ tới server Flask chạy trên máy cục bộ. Tất cả các request sau đó đều dựa trên base URL này để gọi tới các endpoint /api/player và /api/club.

- Dùng thư viện argparse để phân tích tham số dòng lệnh. Chương trình hỗ trợ hai tham số:

- --name <tên cầu thủ>
- --club <tên câu lạc bộ>

Nếu người dùng không truyền vào bất kỳ tham số nào, chương trình in hướng dẫn sử dụng (help) và báo lỗi yêu cầu phải cung cấp ít nhất một trong hai lựa chọn.

- Hàm query_player(player_name) dùng requests.get để gửi yêu cầu HTTP đến endpoint /api/player/<player_name>. Hàm xử lý các trường hợp:

- Nếu status code = 200: parse JSON và trả dữ liệu về cho hàm main.
- Nếu status code = 404: in thông báo lỗi từ server (ví dụ "Player not found") và gợi ý (suggestion) nếu có.
- Nếu status code khác: in mã lỗi HTTP chung.
- Nếu gặp lỗi kết nối (ConnectionError): in thông báo không kết nối được API và nhắc người dùng kiểm tra xem server Flask đã được chạy chưa.

Tương tự, hàm query_club(club_name) gửi request tới /api/club/<club_name> và xử lý lỗi 404 bằng cách in thêm danh sách các câu lạc bộ hiện có (available_clubs) do API trả về, giúp người dùng chọn đúng tên.

- Để hiển thị dữ liệu đẹp hơn, em sử dụng pandas kết hợp với thư viện tabulate. Hàm format_table_display nhận vào một list các dict (danh sách cầu thủ),

chuyển sang DataFrame và chọn ra một số cột quan trọng để ưu tiên hiển thị như Player, Team, Pos, Age, Standard_Min, Standard_Gls, Standard_Ast, Standard_xG, Standard_xAG, Transfer_Value_2024_25. Nếu còn chỗ (max_cols), hàm sẽ bổ sung thêm một số cột còn lại vào bảng hiển thị.

- Hàm display_player_data xử lý hai trường hợp mà API có thể trả về: một cầu thủ duy nhất (trường 'player') hoặc nhiều cầu thủ (trường 'players' trong trường hợp so khớp một phần tên). Với trường hợp một cầu thủ duy nhất, chương trình in thông tin dạng dọc (mỗi dòng một thuộc tính: tên cột và giá trị). Với nhiều cầu thủ, chương trình gọi format_table_display để chọn cột rồi dùng tabulate in bảng dạng lưới (grid) cho dễ nhìn.

- Hàm display_club_data cũng tương tự nhưng luôn làm việc với một danh sách cầu thủ của một câu lạc bộ. Trước khi in bảng, hàm in ra tên câu lạc bộ và tổng số cầu thủ mà API trả về.

- Sau khi hiển thị dữ liệu, chương trình cần lưu kết quả ra file CSV. Hàm save_to_csv nhận DataFrame và tên file, tự động ghép với thư mục chứa file code để lưu vào đúng nơi. Hàm in ra đường dẫn file, số hàng và số cột được lưu để người dùng kiểm tra.

- Để đảm bảo tên file hợp lệ (không chứa ký tự cấm trong tên file của Windows), em cài đặt hàm sanitize_filename. Hàm này thay khoảng trắng bằng dấu gạch dưới và loại bỏ các ký tự đặc biệt như < > : " / \ | ? *. Nhờ đó, tên file xuất ra sẽ an toàn, ví dụ "Mohamed Salah" → "Mohamed_Salah_stats.csv" khi tra cứu theo cầu thủ hoặc "Liverpool" → "Liverpool_players.csv" khi tra cứu theo câu lạc bộ.

- Trong hàm main, chương trình in phần tiêu đề, sau đó dựa trên tham số dòng lệnh để quyết định gọi query_player hay query_club. Kết quả JSON trả về từ API được chuyển thành DataFrame bằng các hàm display_* tương ứng. Nếu có DataFrame và tên file phù hợp, chương trình gọi save_to_csv để lưu ra file CSV; nếu không có dữ liệu (ví dụ nhập sai tên hoặc server trả lỗi), chương trình thông báo "No data to save" và kết thúc.

Như vậy, phần II.2 đã hoàn thành yêu cầu: xây dựng một chương trình dòng lệnh sử dụng requests để gọi API Flask, hiển thị kết quả tra cứu dạng bảng và đồng thời xuất dữ liệu ra file CSV với tên tương ứng với tham số truy vấn.

2. Thuật toán áp dụng

Thuật toán của chương trình lookup.py ở phần II.2 có thể mô tả theo các bước sau:

Bước 1: Phân tích tham số dòng lệnh

- Khởi tạo `argparse.ArgumentParser` với phần mô tả và ví dụ minh họa cách sử dụng.
- Thêm hai tham số:
 - `--name`: tên cầu thủ cần tra cứu (kiểu chuỗi).
 - `--club`: tên câu lạc bộ cần tra cứu (kiểu chuỗi).
- Gọi `parser.parse_args()` để lấy đối tượng `args` chứa các tham số người dùng truyền vào.
- Nếu cả `args.name` và `args.club` đều không có giá trị, in help và báo lỗi yêu cầu truyền ít nhất một tham số, sau đó thoát chương trình.

Bước 2: Xử lý logic tra cứu

- In ra dòng phân cách và tiêu đề "Premier League Player Statistics Lookup".
- Khởi tạo `result_df = None` và `output_filename = None`.
- Nếu `args.name` có giá trị (tra cứu theo cầu thủ):
 - Gọi `query_player(args.name)` để gửi request tới API `/api/player/<tên_cầu_thủ>`.
 - Nếu dữ liệu trả về hợp lệ và có trường `success=True`, gọi `display_player_data(data)` để in dữ liệu ra màn hình dưới dạng bảng và nhận về một `DataFrame` chứa dữ liệu đầy đủ. Đặt `output_filename = "<tên_cầu_thủ>_stats.csv"` sau khi xử lý qua `sanitize_filename`.
- Ngược lại, nếu `args.club` có giá trị (tra cứu theo câu lạc bộ):
 - Gọi `query_club(args.club)` để gửi request tới API `/api/club/<tên_club>`.

- Nếu dữ liệu trả về hợp lệ và `success=True`, gọi `display_club_data(data)` để in bảng danh sách cầu thủ trong đội và nhận về DataFrame. Đặt `output_filename = "<tên_club>_players.csv"` sau khi sanitize.

Bước 3: Gửi request tới API (query_player và query_club)

- Cả hai hàm đều xây dựng URL từ `API_BASE_URL` kết hợp với đường dẫn endpoint tương ứng.
- Dùng `requests.get` với `timeout=10` để gửi request.
- Nếu status code = 200: trả về `response.json()` cho hàm main.
- Nếu status code = 404: đọc thông báo lỗi từ JSON, in ra message và các gợi ý (suggestion hoặc available_clubs), trả về None.
- Nếu status code khác: in mã lỗi HTTP chung và trả về None.
- Bắt các ngoại lệ như `ConnectionError` (không kết nối được API), `Timeout` (quá thời gian chờ) và `Exception` chung, in thông báo lỗi tương ứng cho người dùng.

Bước 4: Hiển thị dữ liệu dạng bảng

- `display_player_data(data)`:

- Nếu data chứa key 'player': nghĩa là API trả về duy nhất một cầu thủ. Hàm tạo DataFrame 1 dòng từ dict này và in từng cột dưới dạng "tên_cột: giá trị" để thể hiện chi tiết.

- Nếu data chứa key 'players': nghĩa là có nhiều cầu thủ (kết quả partial match). Hàm gọi `format_table_display` để chọn cột quan trọng và in ra bảng dạng grid bằng `tabulate`.

- `display_club_data(data)`:

- Giả định data luôn chứa key 'players'. Hàm in tên câu lạc bộ, tổng số cầu thủ và sau đó in bảng danh sách cầu thủ bằng `tabulate`.

- `format_table_display(data, max_cols)`:

- Chuyển list dict sang DataFrame.
- Tạo danh sách các cột ưu tiên (Player, Team, Pos, Age, Standard_Min, Standard_Gls, Standard_Ast, Standard_xG, Standard_xAG, Transfer_Value_2024_25).
- Giữ lại các cột ưu tiên nào thực sự tồn tại trong DataFrame.
- Nếu vẫn còn chỗ (ít hơn max_cols cột), bổ sung thêm một số cột còn lại vào danh sách hiển thị.
- Trả về DataFrame chỉ chứa các cột đã chọn.

Bước 5: Lưu kết quả ra file CSV

- Nếu sau khi tra cứu, result_df không None và output_filename có giá trị:
 - Gọi save_to_csv(result_df, output_filename).
 - Hàm save_to_csv ghép đường dẫn thư mục script với tên file, gọi df.to_csv(..., index=False, encoding='utf-8') để ghi xuống đĩa.
 - In ra đường dẫn file, số hàng và số cột để người dùng kiểm tra.
- Nếu không có dữ liệu (kết quả None): in thông báo "No data to save" và kết thúc chương trình với mã lỗi.

Bước 6: Chuẩn hóa tên file (sanitize_filename)

- Thay khoảng trắng bằng dấu gạch dưới.
- Loại bỏ các ký tự không hợp lệ trong tên file như < > : " / \ | ? *.
- Trả lại chuỗi đã được xử lý để dùng làm tên file CSV an toàn trên hệ điều hành.

Kết quả chạy được:

```
Terminal Local (2) x Local x + v
# No data to save

(.venv) PS C:\Users\Admin\IdeaProjects\BTL-Python\main> python lookup.py --club "Chelsea"
=====
Premier League Player Statistics Lookup
=====

Searching for club: Chelsea...

Club: Chelsea
Total Players: 26
=====

| Player | Team | Pos | Age | Standard_Min | Standard_Glo | Standard_Ast | Standard_LG | Standard_XG | Transfer_Value_2024_25 | Defense_Att | Defense_Blocks |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Axel Disasi | Chelsea | DF | 26 | 364 | 1 | 0 | 0.3 | 0.1 | N/A | 11 | 4 |
| Benoit Badashile | Chelsea | DF | 23 | 334 | 0 | 0 | 0 | 0 | N/A | 5 | 2 |
| Christopher Nwankwu | Chelsea | FW | 26 | 921 | 3 | 2 | 4.8 | 2.1 | €36.3M | 8 | 9 |
| Cole Palmer | Chelsea | MF,FW | 22 | 3191 | 15 | 8 | 17.3 | 10.9 | €111M | 27 | 16 |
| Enzo Fernández | Chelsea | MF | 23 | 2947 | 6 | 7 | 6.1 | 4.8 | N/A | 84 | 43 |
| Filip Jørgensen | Chelsea | GK | 22 | 540 | 0 | 0 | 0 | 0 | N/A | 0 | 0 |
| Jadon Sancho | Chelsea | FW | 24 | 1764 | 3 | 4 | 2.2 | 3.7 | €20.8M | 12 | 11 |
| Joshua Acheampong | Chelsea | DF | 18 | 170 | 0 | 0 | 0.2 | 0 | €35.8M | 2 | 1 |
| Jolo Felix | Chelsea | FW,MF | 24 | 369 | 1 | 1 | 2.2 | 1 | N/A | 9 | 2 |
| Kiernan Dewsbury-Hall | Chelsea | MF,FW | 25 | 264 | 0 | 1 | 0.2 | 1.1 | N/A | 6 | 2 |
| Levi Colwill | Chelsea | DF | 21 | 3249 | 2 | 1 | 1.9 | 1.1 | N/A | 27 | 37 |
| Malo Gusto | Chelsea | DF | 21 | 1862 | 0 | 1 | 1.7 | 1.4 | €32.1M | 48 | 29 |
| Marc Cucurella | Chelsea | DF | 26 | 2988 | 5 | 1 | 2.8 | 2.1 | €45.0M | 61 | 49 |
| Moisés Caicedo | Chelsea | MF,DF | 22 | 3351 | 1 | 2 | 0.8 | 3 | €100.0M | 86 | 49 |
| Mykhailo Mudryk | Chelsea | FW | 23 | 151 | 0 | 0 | 0.3 | 1 | N/A | 0 | 3 |
| Nicolas Jackson | Chelsea | FW | 23 | 2220 | 10 | 5 | 12.3 | 4.3 | €50.0M | 10 | 15 |

| Marc Cucurella | Chelsea | DF | 26 | 2988 | 5 | 1 | 2.8 | 2.1 | €45.0M | 61 | 49 |
| Moisés Caicedo | Chelsea | MF,DF | 22 | 3351 | 1 | 2 | 0.8 | 3 | €100.0M | 86 | 49 |
| Mykhailo Mudryk | Chelsea | FW | 23 | 151 | 0 | 0 | 0.3 | 1 | N/A | 0 | 3 |
| Nicolas Jackson | Chelsea | FW | 23 | 2220 | 10 | 5 | 12.3 | 4.3 | €50.0M | 10 | 15 |
| Noni Madueke | Chelsea | FW | 22 | 2033 | 7 | 3 | 9.6 | 4.2 | N/A | 38 | 17 |
| Pedro Neto | Chelsea | FW | 24 | 2269 | 4 | 6 | 3.5 | 6.7 | €55.0M | 21 | 14 |
| Reece James | Chelsea | DF,MF | 24 | 1043 | 1 | 1 | 0.4 | 0.7 | €40.2M | 9 | 14 |
| Renato Veiga | Chelsea | DF,MF | 21 | 181 | 0 | 0 | 0.1 | 0 | €25.0M | 12 | 3 |
| Robert Sánchez | Chelsea | GK | 26 | 2880 | 0 | 0 | 0 | 0.1 | €20.0M | 1 | 0 |
| Roméo Lavia | Chelsea | MF | 20 | 797 | 0 | 1 | 0.3 | 0.7 | €35.0M | 21 | 6 |
| Tosin Adarabioyo | Chelsea | DF | 26 | 1409 | 1 | 1 | 0.9 | 0.2 | €23.6M | 12 | 9 |
| Trevoh Chalobah | Chelsea | DF | 25 | 910 | 0 | 1 | 0.8 | 1.2 | €41.7M | 8 | 6 |
| Tyrrique George | Chelsea | FW | 18 | 183 | 1 | 1 | 0.2 | 0.7 | €24.4M | 3 | 1 |
| Wesley Fofana | Chelsea | DF | 23 | 1172 | 0 | 0 | 0.4 | 0.4 | €27.5M | 14 | 14 |

Date saved to: C:\Users\Admin\IdeaProjects\BTL-Python\main\Chelsea.Players.csv
Total rows: 26
Total columns: 68

(.venv) PS C:\Users\Admin\IdeaProjects\BTL-Python\main>
```

Age	Defense_J	Defense_I	Defense_5	Defense_1	Defense_5	Defense_2	Defense_1	Goalkeeper	Goalkeeper	Goalkeeper	Misc_Crs	Misc_Fld	Misc_Rls	Misc_Lost	Misc_Off	Misc_Recc	Misc_Won	Misc_Nation	Passing_1	Passing_Cr	Passing_Cr	Passing_Kr	Passing_M	Passing_Pf	Passing_Sf	Passing_Tc	Pf				
26	11	4	2	5	3	1	7	4	N/A	N/A	N/A	0	0	2	2	0	15	5	71.4	FRA	20	266	0	2	69.2	92.2	2	94.4	4218	90.8	
23	5	2	6	1	0	2	9	6	N/A	N/A	N/A	0	2	6	2	0	18	8	80	FRA	37	273	0	0	65.6	93.8	1	93.3	4820	89.8	
26	8	9	7	2	7	2	12	8	N/A	N/A	N/A	4	11	16	22	3	29	16	42.1	FRA	26	299	0	14	100	92.2	7	91.6	3928	89.5	
22	27	16	11	15	15	1	34	20	N/A	N/A	N/A	144	60	15	7	7	106	2	22.2	ENG	163	1140	17	87	59.9	84.7	64	86.1	21582	77	
23	84	43	13	53	36	7	65	33	N/A	N/A	N/A	109	54	52	29	2	156	14	32.6	ARG	167	1518	7	77	57.5	84.9	41	88.7	26776	79.9	
22	0	0	0	0	0	0	0	0	16.7	1.5	71.4	0	1	1	0	0	5	0	N/A	DEF	1	158	0	0	27.5	97.5	0	100	2843	83.6	
24	12	11	2	7	11	0	13	7	N/A	N/A	N/A	25	11	6	11	6	99	6	35.3	ENG	28	663	0	38	51.9	83.2	48	89.5	8932	82.6	
18	2	1	1	0	1	0	2	1	N/A	N/A	N/A	0	1	0	6	0	7	1	14.3	ENG	6	104	0	0	55	92.6	0	90.6	2025	83.9	
24	9	2	3	4	2	0	8	3	N/A	N/A	N/A	0	17	6	5	1	22	4	44.4	POR	22	171	0	6	66.7	91.2	8	90.5	2588	85.9	
25	6	2	2	2	2	0	8	5	N/A	N/A	N/A	2	6	4	1	0	9	0	0	ENG	12	127	0	3	50	83.9	3	91.5	1898	83.6	
21	27	37	33	9	12	25	47	34	N/A	N/A	N/A	13	21	45	53	5	121	82	60.7	ENG	148	2338	3	14	56.1	95.2	7	92.4	42476	89.3	
21	48	29	24	23	21	8	45	29	N/A	N/A	N/A	53	20	20	15	2	119	9	37.5	FRA	111	1198	7	22	60.7	88.4	19	93.2	18538	86.5	
26	61	49	28	29	33	16	67	45	N/A	N/A	N/A	31	38	37	57	3	130	32	36	ESP	121	1680	3	19	56.1	89.4	12	94	26450	87.7	
22	86	49	49	32	39	10	114	73	N/A	N/A	N/A	10	58	70	16	0	229	35	48.6	ECU	235	1967	2	30	76.4	92.3	34	92.6	33834	89.7	
23	0	3	0	0	2	1	2	2	N/A	N/A	N/A	7	2	2	4	1	9	2	33.3	UKR	2	28	1	2	50	66.7	2	70.8	431	66.7	
23	10	15	3	5	13	2	22	13	N/A	N/A	N/A	3	29	35	36	23	48	21	36.8	SEN	22	303	1	28	60	78	10	81.6	4088	75.9	
22	38	17	10	24	17	0	25	18	N/A	N/A	N/A	63	25	22	19	5	84	11	36.7	ENG	16	530	8	32	53.7	76.3	35	86.2	7705	77.4	
24	21	14	9	12	12	2	33	9	N/A	N/A	N/A	149	33	21	30	3	81	11	26.8	POR	30	702	23	46	47.2	81.9	37	93	11347	78.4	
24	9	14	13	6	8	6	10	9	N/A	N/A	N/A	43	11	8	9	0	66	23	71.9	ENG	80	717	1	11	49.1	89.3	5	93.3	12618	84	
21	12	3	1	8	2	1	9	6	N/A	N/A	N/A	0	1	6	3	1	9	3	50	POR	6	118	0	0	61.9	90.2	1	88.2	2171	84.9	
26	1	0	2	1	0	0	0	0	31.3	1.06	78.4	0	4	2	0	0	43	12	100	ESP	26	871	0	2	36.3	96.6	1	99.5	21022	89.7	
20	21	6	16	11	6	0	22	13	N/A	N/A	N/A	0	7	16	6	0	40	12	66.7	BEL	49	389	0	7	84	94.8	3	91.8	6552	91.7	
26	12	9	11	4	2	7	17	13	N/A	N/A	N/A	0	9	8	28	1	41	42	60	ENG	62	1079	0	1	60.6	94.8	1	95.5	19805	91.1	
25	8	6	10	1	3	0	3	11	7	N/A	N/A	N/A	3	7	9	12	0	29	23	65.7	ENG	42	609	0	3	52.8	95.5	3	95	10527	90
18	3	1	2	1	1	0	2	0	N/A	N/A	N/A	3	0	3	4	2	4	0	0	ENG	0	43	0	6	0	84.2	1	88.9	551	79.6	
23	14	14	14	8	5	9	13	12	N/A	N/A	N/A	1	16	24	12	0	54	27	69.2	FRA	54	724	0	6	61.9	92.5	1	93.9	12262	90.2	

III.1.1 – Tính trung vị, trung bình và độ lệch chuẩn cho từng đội

1. Lý giải cách làm

Ở mục III.1.1, yêu cầu là:

- Với mỗi đội bóng, tính trung vị (median), trung bình (mean) và độ lệch chuẩn (standard deviation) của các chỉ số cầu thủ.
- Ghi kết quả ra 1 file CSV để dùng tiếp cho các phân tích sau.

Code nhóm em viết trong file `problem3_1_copy.py` thực hiện như sau:

- Đầu tiên, chương trình dùng thư viện `os` để xác định đường dẫn thư mục chứa file code (`script_dir`), sau đó ghép với tên file `'results.csv'` để tạo đường dẫn đầy đủ tới dữ liệu đầu vào (`input_path`). Đây chính là file tổng hợp tất cả các chỉ số cầu thủ thu được ở phần I.
- Việc đọc file CSV được đặt trong khối `try/except` để xử lý lỗi: nếu file không tồn tại thì `raise FileNotFoundError` và in thông báo; nếu file rỗng hoặc bị lỗi định dạng, chương trình báo lỗi tương ứng và `exit(1)`. Nếu đọc thành công, `pandas.read_csv` trả về `DataFrame` `data` và in ra thông báo "Successfully loaded data".
- Sau khi có `DataFrame`, chương trình cần xác định các cột nào là "chỉ số" cần tính thống kê. Điều này được thực hiện bằng lệnh `data.select_dtypes(include=[float, int]).columns` để lấy ra tất cả các cột có kiểu dữ liệu số (float hoặc int). Danh sách này được lưu trong biến `number_attributes` và chính là tập các chỉ số sẽ được tính median/mean/std.
- Kết quả thống kê sẽ được chứa trong một từ điển Python tên là `results`. Ban đầu, `results` chỉ có 1 key là `'Team'` với giá trị là một list có phần tử đầu tiên là `'all'`. Hàng `'all'` đại diện cho thống kê chung cho toàn bộ giải (không phân biệt đội bóng).
- Tiếp theo, chương trình duyệt qua từng cột chỉ số `x` trong `number_attributes` và tính lần lượt:
 - Trung vị trên toàn bộ `data[x]` bằng `data[x].median(skipna=True)`.
 - Trung bình bằng `data[x].mean(skipna=True)`.
 - Độ lệch chuẩn bằng `data[x].std(skipna=True)`.Các giá trị này được format về chuỗi với 2 chữ số thập phân bằng `f"{...:.2f}"` rồi đưa vào từ điển `results` với các key dạng `'Median of x'`, `'Mean of x'`, `'Std of x'`. Ở thời điểm này, mỗi key tương ứng là một list chứa giá trị thống kê cho hàng `'all'`.
- Sau khi có thống kê chung cho toàn giải, chương trình chuyển sang tính thống kê theo từng đội bóng. Dữ liệu được group theo cột `'Team'` bằng

`data.groupby('Team')`. Với mỗi cặp (team, group) thu được:

- Thêm tên đội vào cột 'Team' của results bằng `results['Team'].append(team)`.
- Với từng chỉ số x trong `number_attributes`, tính median/mean/std của `group[x]` (chỉ tính trên các cầu thủ thuộc đội đó) rồi append thêm vào các list tương ứng trong results: 'Median of x', 'Mean of x', 'Std of x'.
- Kết quả là từ điển results có cấu trúc: mỗi key là một cột (Team, Median of ..., Mean of ..., Std of ...), mỗi value là một list, trong đó phần tử đầu tiên là thống kê cho toàn giải ('all'), các phần tử tiếp theo lần lượt là thống kê cho từng đội bóng.
- Cuối cùng, chương trình chuyển results thành DataFrame `results_df` bằng `pandas.DataFrame(results)` rồi lưu ra file 'results2.csv' bằng `results_df.to_csv('results2.csv', index=False)`. File này là đầu ra của mục III.1.1, đồng thời là đầu vào để tiếp tục phân tích ở các phần sau (tìm đội dẫn đầu từng chỉ số, xếp hạng đội...).

Sau này khi viết báo cáo, em có thể chèn thêm bảng minh họa lấy từ một phần dữ liệu trong results2.csv để thể hiện kết quả chạy được cho vài đội bóng tiêu biểu.

2. Thuật toán áp dụng

Thuật toán cho phần III.1.1 (file `problem3_1_copy.py`) có thể mô tả ngắn gọn như sau:

Bước 1: Đọc dữ liệu đầu vào

- Xác định đường dẫn tới file results.csv.
- Dùng `pandas.read_csv` đọc dữ liệu vào DataFrame data trong khối try/except để xử lý các lỗi về file.

Bước 2: Xác định các cột chỉ số cần tính

- Dùng `data.select_dtypes(include=[float, int]).columns` để lấy danh sách các cột kiểu số, đặt tên là `number_attributes`.

Bước 3: Tính median/mean/std cho toàn giải

- Khởi tạo từ điển results với key 'Team' và giá trị ban đầu ['all'].
- Với mỗi cột x trong `number_attributes`:
 - Tính `median_x = median(data[x])` với `skipna=True`.

- Tính `mean_x = mean(data[x])` với `skipna=True`.
- Tính `std_x = std(data[x])` với `skipna=True`.
- Thêm các giá trị đã format 2 chữ số thập phân vào các list `results['Median of x']`, `results['Mean of x']`, `results['Std of x']`.

Bước 4: Tính median/mean/std cho từng đội bóng

- Group dữ liệu theo cột 'Team': `for team, group in data.groupby('Team'):`
 - Thêm tên đội vào `results['Team']`.
 - Với mỗi cột x trong `number_attributes`:
 - + Tính `median_x_team = median(group[x])`.
 - + Tính `mean_x_team = mean(group[x])`.
 - + Tính `std_x_team = std(group[x])`.
 - + Append các giá trị này (đã format) vào list tương ứng trong `results`.

Bước 5: Xuất kết quả ra file CSV

- Tạo DataFrame `results_df = pd.DataFrame(results)`.
- Ghi `results_df` ra file 'results2.csv' với `index=False`.

3. Kết quả chạy được

```
PS D:\VIEnguyen72 Downloads\BTL-Python-main> & C:\Users\Admin\AppData\Local\Programs\Python\Python313\python.exe "d:/VIEnguyen72 Downloads/BTL-Python-main/BTL-Python-main/main/pro3.py"
Successfully loaded data from d:\VIEnguyen72 Downloads\BTL-Python-main\BTL-Python-main\main\results2.csv
Tất cả các cột cần thiết đều tồn tại: 17 cột
Median of Standard_Gls: Arsenal (giá trị: 2.5)
Mean of Standard_Gls: Liverpool (giá trị: 3.86)
Median of Standard_Ast: Arsenal (giá trị: 2.0)
Mean of Standard_Ast: Liverpool (giá trị: 2.95)
Median of Standard_xG: Arsenal (giá trị: 1.85)
Mean of Standard_xG: Liverpool (giá trị: 3.82)
Median of Standard_xAG: Crystal Palace (giá trị: 1.65)
Mean of Standard_xAG: Liverpool (giá trị: 2.83)
Median of Passing_Cmp: Tottenham (giá trị: 716.0)
Mean of Passing_Cmp: Manchester City (giá trị: 857.88)
Median of Possession_Touches: Crystal Palace (giá trị: 1014.0)
Mean of Possession_Touches: Liverpool (giá trị: 1176.95)
Median of Defense_Int: Crystal Palace (giá trị: 11.0)
Mean of Defense_Int: Everton (giá trị: 15.64)
Median of Goalkeeping_Save%: Bournemouth (giá trị: 88.0)
Mean of Goalkeeping_Save%: Bournemouth (giá trị: 79.5)

Tìm thấy 16 chỉ số có đội dẫn đầu
Thống kê: 16/16 chỉ số xử lý thành công

=====
BẢNG XẾP HẠNG ĐỘI DẪN ĐẦU THEO CHỈ SỐ
=====
1. Liverpool: 5 chỉ số
2. Arsenal: 3 chỉ số
3. Crystal Palace: 3 chỉ số
4. Bournemouth: 2 chỉ số
5. Tottenham: 1 chỉ số
6. Manchester City: 1 chỉ số
7. Everton: 1 chỉ số

Tổng số đội dẫn đầu: 7
Tổng chỉ số đã xét: 16
PS D:\VIEnguyen72 Downloads\BTL-Python-main>
```

III.1.2 – Tìm đội bóng có chỉ số cao nhất ở mỗi chỉ số

1. Lý giải cách làm

Ở mục III.1.2, yêu cầu là:

- Từ bảng thống kê đã có (trung vị, trung bình, độ lệch chuẩn của từng chỉ số cho mỗi đội ở mục III.1.1), tìm đội bóng có "điểm số" cao nhất ở từng chỉ số.
- Từ đó rút ra nhận xét đội nào có phong độ tốt nhất giải Ngoại hạng Anh mùa 2024–2025.

Để thực hiện, nhóm đã viết chương trình trong file `problem3_1_2_copy.py`, sử dụng đầu vào là file `results2.csv` được tạo ở bước III.1.1.

- Đầu tiên, chương trình sử dụng `os.path` để xác định đường dẫn tới file `results2.csv` (cùng thư mục với code), sau đó đọc file bằng `pandas.read_csv`. Phần đọc file được đặt trong khối `try/except` để xử lý các lỗi thường gặp như: file không tồn tại, file rỗng hoặc lỗi parse.

- DataFrame `df` này chứa một dòng "all" (thống kê chung) và nhiều dòng còn lại tương ứng với từng đội bóng. Mỗi cột là một thống kê

- Median of <tên chỉ số>
- Mean of <tên chỉ số>
- Std of <tên chỉ số>

trong đó em chỉ sử dụng các cột Median và Mean cho việc đánh giá.

- Nhóm định nghĩa một danh sách `chi_so_quan_trong` gồm các cột thống kê quan trọng muốn dùng để so sánh sức mạnh các đội, ví dụ: Median/Mean của `Standard_Gls` (bàn thắng), `Standard_Ast` (kiến tạo), `Standard_xG`, `Standard_xAG`, `Passing_Cmp` (chuyền bóng thành công), `Possession_Touches` (số lần chạm bóng), `Defense_Int` (cắt bóng), `Goalkeeping_Save%` (tỉ lệ cứu thua của thủ môn). Danh sách này bao phủ cả khâu tấn công, kiến tạo, kiểm soát bóng, phòng ngự và thủ môn.

- Đối với mỗi chỉ số trong `chi_so_quan_trong`, chương trình tìm đội có giá trị cao nhất bằng cách dùng hàm `idxmax()` của `pandas`. Cụ thể:

- `max_index = df[chi_so].idxmax()` cho biết dòng có giá trị lớn nhất của cột `chi_so`.

- `doi_tot_nhat = df.loc[max_index, 'Team']` là tên đội tương ứng.

- `gia_tri = df.loc[max_index, chi_so]` là giá trị thống kê lớn nhất của chỉ số đó.

Các thông tin này được lưu vào hai từ điển: `doi_dan_dau[chi_so] =`

`doi_tot_nhat` và `gia_tri_cao_nhat[chi_so] = gia_tri`.

- Sau khi xác định đội dẫn đầu cho từng chỉ số, em cần tổng hợp lại để xem đội

nào dẫn đầu nhiều chỉ số nhất. Chương trình khai báo hai từ điển:

- `diem_so_doi`: lưu số lượng chỉ số mà mỗi đội dẫn đầu.
- `chi_so_cua_doi`: lưu danh sách các chỉ số mà đội đó đứng đầu.

Khi duyệt qua từng phần tử (`chi_so`, `doi`) trong `doi_dan_dau`, nếu đội đã có trong `diem_so_doi` thì tăng thêm 1 và append `chi_so` vào `chi_so_cua_doi[doi]`; nếu chưa có thì khởi tạo điểm số = 1 và danh sách chỉ số là `[chi_so]`.

- Sau khi đếm xong, chương trình sắp xếp các đội theo số chỉ số dẫn đầu (giảm dần) bằng hàm `sorted` với key là giá trị điểm số. Kết quả danh sách sắp xếp này (`doi_xep_hang`) chính là bảng xếp hạng các đội về "độ phủ" chỉ số – đội nào dẫn đầu càng nhiều chỉ số thì được coi là có phong độ tổng thể càng tốt.

- Cuối cùng, chương trình in ra hai phần kết quả:

- Phần 1: liệt kê từng chỉ số, đội dẫn đầu và giá trị thống kê cụ thể (đã format 1 chữ số thập phân).
- Phần 2: bảng xếp hạng đội dẫn đầu theo chỉ số, trong đó mỗi dòng gồm: thứ hạng, tên đội, tổng số chỉ số dẫn đầu và danh sách từng chỉ số cụ thể mà đội đó đứng đầu.

2. Thuật toán áp dụng

Bước 1: Đọc dữ liệu từ file `results2.csv`

- Xác định đường dẫn thư mục chứa file code (`script_dir`) và ghép với tên `'results2.csv'` để có `input_path`.
- Kiểm tra nếu file không tồn tại thì báo lỗi `FileNotFoundError` và kết thúc chương trình.
- Nếu file tồn tại, dùng `pandas.read_csv` đọc vào `DataFrame df`, kèm các khối `try/except` để xử lý lỗi file rỗng hoặc lỗi parser.

Bước 2: Khai báo danh sách các chỉ số quan trọng

- Tạo list `chi_so_quan_trong` gồm các tên cột thống kê dạng `'Median of ...'` và `'Mean of ...'` cho các chỉ số tấn công, kiến tạo, xG/xAG, chuyển bóng, kiểm soát bóng, phòng ngự, thủ môn.

Bước 3: Tìm đội dẫn đầu cho từng chỉ số

- Khởi tạo hai từ điển rỗng: `doi_dan_dau` và `gia_tri_cao_nhat`.
- Với mỗi `chi_so` trong `chi_so_quan_trong`:
 - Tìm `max_index = df[chi_so].idxmax()` để lấy index của dòng có giá trị cao nhất.
 - Lấy `doi_tot_nhat = df.loc[max_index, 'Team']`.

- Lấy gia_tri = df.loc[max_index, chi_so].
- Gán doi_dan_dau[chi_so] = doi_tot_nhat và gia_tri_cao_nhat[chi_so] = gia_tri.

Bước 4: Đếm điểm cho từng đội và lưu các chỉ số dẫn đầu

- Khởi tạo hai từ điển: diem_so_doi = {} và chi_so_cua_doi = {}.
- Duyệt qua từng cặp (chi_so, doi) trong doi_dan_dau.items():
 - Nếu đội đã có trong diem_so_doi:
 - + Tăng diem_so_doi[doi] thêm 1.
 - + Append chi_so vào danh sách chi_so_cua_doi[doi].
 - Nếu đội chưa xuất hiện:
 - + Gán diem_so_doi[doi] = 1.
 - + Gán chi_so_cua_doi[doi] = [chi_so].

Bước 5: Sắp xếp các đội theo điểm số

- Tạo danh sách doi_xep_hang bằng cách sắp xếp diem_so_doi.items() theo giá trị điểm số giảm dần:

```
doi_xep_hang = sorted(diem_so_doi.items(), key=lambda x: x[1],
reverse=True).
```

Bước 6: In kết quả chi tiết

- In phần tiêu đề "ĐỘI DẪN ĐẦU TỪNG CHỈ SỐ VÀ GIÁ TRỊ CỤ THỂ", sau đó duyệt qua doi_dan_dau:

- Với mỗi chi_so, lấy doi = doi_dan_dau[chi_so] và gia_tri = gia_tri_cao_nhat[chi_so].

- In tên chỉ số, đội dẫn đầu và giá trị thống kê (format một chữ số thập phân).

- In phần tiêu đề "BẢNG XẾP HẠNG ĐỘI DẪN ĐẦU THEO CHỈ SỐ", sau đó duyệt qua danh sách doi_xep_hang với i là thứ hạng, doi là tên đội, diem là số chỉ số dẫn đầu:

- In thứ hạng, tên đội và số chỉ số đội đó dẫn đầu.
- Dùng chi_so_cua_doi[doi] để in danh sách các chỉ số cụ thể mà đội này dẫn đầu, kèm giá trị tương ứng từ gia_tri_cao_nhat.

3. Kết quả chạy được

```

PS D:\VIEnguyen72 Downloads\BTL-Python-main> & C:\Users\Admin\AppData\Local\Programs\Python\Python313\python.exe "d:\VIEnguyen72 Downloads\BTL-Python-main\BTL-Pyth
-main/main/problem3_1_2_copy.py"
Successfully loaded data from d:\VIEnguyen72 Downloads\BTL-Python-main\BTL-Python-main\main\results2.csv

=====
ĐỘI DẪN ĐẦU TỪNG CHỈ SỐ VÀ GIÁ TRỊ CỤ THỂ
=====
Median of Standard_Gls:
Đội dẫn đầu: Arsenal
Giá trị: 2.5

Mean of Standard_Gls:
Đội dẫn đầu: Liverpool
Giá trị: 3.9

Median of Standard_Ast:
Đội dẫn đầu: Arsenal
Giá trị: 2.0

Mean of Standard_Ast:
Đội dẫn đầu: Liverpool
Giá trị: 3.0

Median of Standard_xG:
Đội dẫn đầu: Arsenal
Giá trị: 1.9

Mean of Standard_xG:
Đội dẫn đầu: Liverpool
Giá trị: 3.8

Median of Standard_xAG:
Đội dẫn đầu: Crystal Palace
Giá trị: 1.6

Mean of Standard_xAG:
Đội dẫn đầu: Liverpool
Giá trị: 2.8

```

```

Mean of Passing_Cmp:
Đội dẫn đầu: Manchester City
Giá trị: 857.9

Median of Possession_Touches:
Đội dẫn đầu: Crystal Palace
Giá trị: 1014.0

Mean of Possession_Touches:
Đội dẫn đầu: Liverpool
Giá trị: 1177.0

Median of Defense_Int:
Đội dẫn đầu: Crystal Palace
Giá trị: 11.0

Mean of Defense_Int:
Đội dẫn đầu: Everton
Giá trị: 15.6

Median of Goalkeeping_Save%:
Đội dẫn đầu: Bournemouth
Giá trị: 80.0

Mean of Goalkeeping_Save%:
Đội dẫn đầu: Bournemouth
Giá trị: 79.5

```

```

=====
BẢNG XẾP HẠNG ĐỘI DẪN ĐẦU THEO CHỈ SỐ
=====
1. Liverpool: 5 chỉ số
Các chỉ số dẫn đầu:
- Mean of Standard_Gls: 3.9
- Mean of Standard_Ast: 3.0
- Mean of Standard_xG: 3.8
- Mean of Standard_xAG: 2.8
- Mean of Possession_Touches: 1177.0

2. Arsenal: 3 chỉ số
Các chỉ số dẫn đầu:
- Median of Standard_Gls: 2.5
- Median of Standard_Ast: 2.0
- Median of Standard_xG: 1.9

3. Crystal Palace: 3 chỉ số
Các chỉ số dẫn đầu:
- Median of Standard_xAG: 1.6
- Median of Possession_Touches: 1014.0
- Median of Defense_Int: 11.0

4. Bournemouth: 2 chỉ số
Các chỉ số dẫn đầu:
- Median of Goalkeeping_Save%: 80.0
- Mean of Goalkeeping_Save%: 79.5

5. Tottenham: 1 chỉ số
Các chỉ số dẫn đầu:
- Median of Passing_Cmp: 716.0

6. Manchester City: 1 chỉ số
Các chỉ số dẫn đầu:
- Mean of Passing_Cmp: 857.9

7. Everton: 1 chỉ số
Các chỉ số dẫn đầu:
- Mean of Defense_Int: 15.6

```

III.2 – Đề xuất phương pháp định giá cầu thủ

1. Lý giải cách làm

Cách làm tổng quát như sau:

- Chương trình bắt đầu bằng việc đọc dữ liệu từ file results.csv – đây là file chứa toàn bộ chỉ số cầu thủ đã được crawl và xử lý ở phần I. Đường dẫn tới file được ghép từ thư mục chứa file code (script_dir) và tên file 'results.csv'. Việc đọc file được đặt trong khối try/except, nếu file không tồn tại hoặc lỗi, chương trình in thông báo tương ứng để tránh bị crash đột ngột.
- Sau khi dữ liệu được nạp vào DataFrame df, chương trình tiến hành xử lý giá trị thiếu (missing values). Trong phần này, các chỉ số quan trọng phục vụ việc định giá (như số bàn thắng, kiến tạo, xG, xAG hoặc số phút thi đấu) được điền giá trị mặc định 0 khi bị thiếu, bởi vì có thể coi là cầu thủ không đóng góp ở các chỉ số đó. Những cột không dùng trực tiếp trong mô hình định giá có thể được bỏ qua hoặc điền giá trị 'N/A' để đánh dấu.
- Phần quan trọng nhất là hàm estimate_transfer_value(row) – đây là hàm nhận vào một dòng dữ liệu tương ứng với một cầu thủ và trả về giá trị chuyển nhượng ước tính (đơn vị triệu euro). Hàm này kết hợp ba yếu tố chính:

1. Giá trị cơ bản theo vị trí (base_value):

- Nếu trong cột Pos có chứa 'FW' (tiền đạo), base_value được đặt cao

nhất.

- Nếu là tiền vệ ('MF'), base_value thấp hơn một chút.
- Nếu là hậu vệ ('DF') hoặc thủ môn ('GK'), base_value thấp hơn nữa.
- Nếu vị trí không rõ, dùng một giá trị mặc định tương đối thấp.

Ý tưởng là phản ánh xu hướng chung của thị trường: các vị trí tấn công thường được định giá cao hơn.

2. Hệ số tuổi (age_factor):

Hàm số tuổi được thiết kế sao cho cầu thủ ở độ tuổi "đẹp" (khoảng 20–25) có hệ số lớn hơn, còn cầu thủ quá già thì hệ số giảm dần nhưng không dưới một ngưỡng tối thiểu. Điều này mô phỏng thực tế: cầu thủ trẻ, còn nhiều tiềm năng phát triển thường được định giá cao hơn cầu thủ đã qua thời kỳ đỉnh cao.

3. Hệ số phong độ (performance_factor):

Hệ số này được tính dựa trên các chỉ số tấn công như số bàn thắng (Standard_Gls), số kiến tạo (Standard_Ast), xG và xAG. Mỗi chỉ số được gán một trọng số phản ánh mức độ quan trọng. Tổng điểm phong độ sau đó được chuẩn hóa (chia cho một hằng số) và cộng thêm 1 để tạo thành hệ số nhân > 1 nếu cầu thủ có phong độ tốt.

- Giá trị chuyển nhượng ước tính cuối cùng được tính bằng công thức:

$$\text{Transfer_Value} = \text{base_value} \times \text{age_factor} \times \text{performance_factor}$$

Như vậy, một cầu thủ ở vị trí tấn công, đang ở độ tuổi đẹp và có phong độ tốt sẽ có Transfer_Value cao hơn.

- Sau khi định nghĩa hàm estimate_transfer_value, chương trình áp dụng hàm này cho toàn bộ DataFrame bằng df.apply(estimate_transfer_value, axis=1) và lưu kết quả vào cột mới 'Transfer_Value'. Một cột nữa là 'Transfer_Value_Str' cũng được tạo ra để hiển thị giá trị ở dạng chuỗi dễ đọc, ví dụ "EUR 42.5M".

- Để minh họa, chương trình in ra top 20 cầu thủ có Transfer_Value cao nhất, kèm theo các thông tin cơ bản như Player, Team, Pos, Age, Standard_Gls, Standard_Ast. Cuối cùng, dữ liệu định giá được chọn lọc các cột quan trọng và lưu vào file results3_2.csv để có thể sử dụng tiếp trong phân tích hoặc báo cáo.

Mặc dù mô hình này đơn giản và chưa xét tới nhiều yếu tố thực tế (hợp đồng, thương hiệu, thể lực,...), nhưng nó thể hiện rõ ý tưởng yêu cầu trong đề: xây dựng một phương pháp định giá dựa trên dữ liệu thống kê đã thu thập được.

2. Thuật toán áp dụng

Bước 1: Đọc dữ liệu và xử lý lỗi

- Xác định đường dẫn tới file results.csv bằng cách lấy thư mục chứa file code (script_dir) rồi ghép với tên file.
- Dùng pandas.read_csv để đọc dữ liệu vào DataFrame df trong khối try/except:
 - Nếu file không tồn tại: báo lỗi FileNotFoundError và in thông báo.
 - Nếu file rỗng hoặc lỗi parser: in thông báo và kết thúc chương trình.

Bước 2: Xử lý giá trị thiếu (missing values)

- Xác định các cột chỉ số quan trọng phục vụ việc định giá (ví dụ: Standard_Gls, Standard_Ast, Standard_xG, Standard_xAG, Standard_Min...).
- Dùng df.fillna(0) hoặc df.fillna với dict để gán giá trị 0 cho các cột định lượng quan trọng bị thiếu.
- Với các cột không sử dụng trực tiếp trong mô hình, có thể gán 'N/A' để tránh lỗi khi in hoặc hiển thị.

Bước 3: Định nghĩa hàm estimate_transfer_value(row)

- Hàm nhận một dòng dữ liệu row tương ứng với một cầu thủ.
- Dựa trên cột Pos, xác định base_value theo vị trí thi đấu (FW, MF, DF, GK hoặc khác).
- Đọc tuổi cầu thủ từ cột Age, tính age_factor theo công thức dạng:
$$\text{age_factor} = \max(0.5, 1.5 - (\text{Age} - 20) * 0.03)$$
để hệ số giảm dần khi tuổi lệch nhiều khỏi mốc 20.
- Lấy các giá trị Gls, Ast, xG, xAG từ các cột Standard_Gls, Standard_Ast, Standard_xG, Standard_xAG; nếu giá trị thiếu thì coi như 0.
- Tính performance_factor theo dạng:
$$\text{performance_factor} = 1 + (\text{Gls} * 0.2 + \text{Ast} * 0.15 + \text{xG} * 0.1 + \text{xAG} * 0.1) / 10$$
để cầu thủ có phong độ tốt sẽ có hệ số lớn hơn 1.
- Trả về giá trị:
$$\text{Transfer_Value} = \text{base_value} * \text{age_factor} * \text{performance_factor}.$$

Bước 4: Áp dụng hàm cho toàn bộ DataFrame

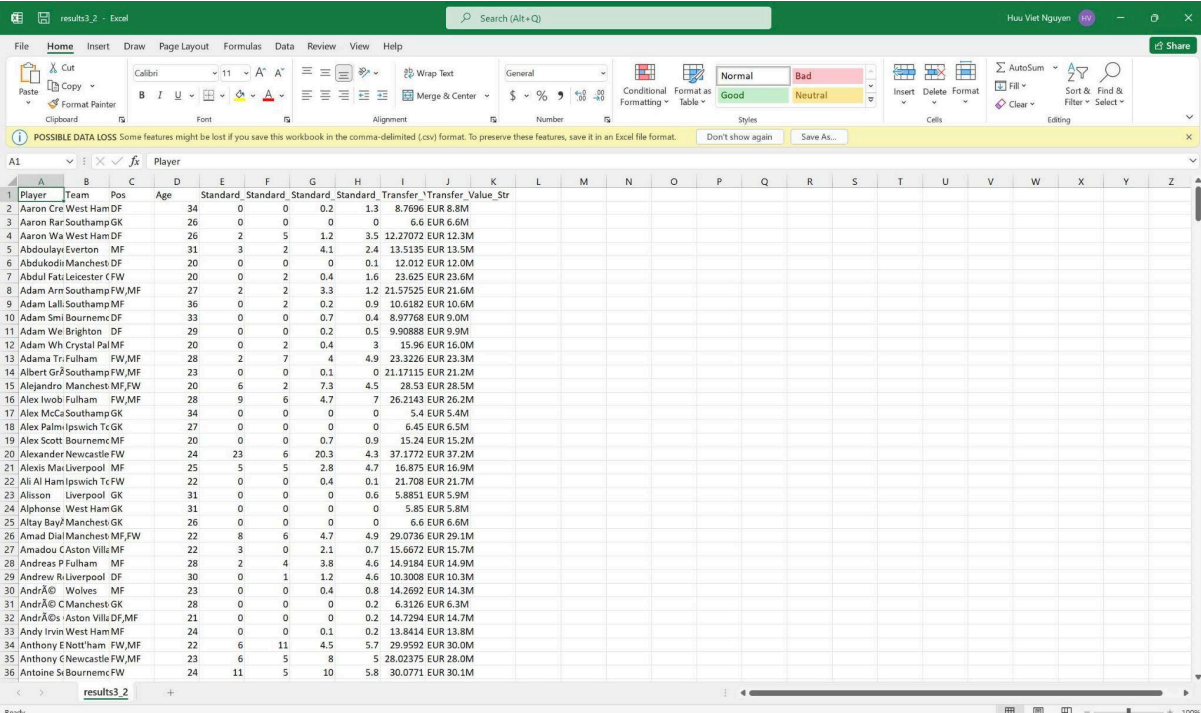
- Gọi df['Transfer_Value'] = df.apply(estimate_transfer_value, axis=1) để tính giá trị chuyển nhượng cho từng cầu thủ.
- Tạo cột df['Transfer_Value_Str'] = df['Transfer_Value'].apply(lambda x: f'EUR {x:.1f}M') để định dạng giá trị dạng chuỗi.

Bước 5: Xuất kết quả và lưu file

-In ra màn hình top 20 cầu thủ có Transfer_Value cao nhất, sắp xếp giảm dần theo cột này, kèm theo các cột Player, Team, Pos, Age, Standard_Gls, Standard_Ast.

-Chọn các cột quan trọng (ví dụ: Player, Team, Pos, Age, Standard_Gls, Standard_Ast, Standard_xG, Standard_xAG, Transfer_Value, Transfer_Value_Str) và lưu ra file results3_2.csv bằng df.to_csv(..., index=False).

3. Kết quả chạy được



Player	Team	Pos	Age	Standard_Gls	Standard_Ast	Standard_xG	Standard_xAG	Transfer_Value	Transfer_Value_Str
Aaron Cresswell	West Ham	DF	34	0	0	0.2	1.3	8.7696	EUR 8.8M
Aaron Ramsdale	Southampton	GK	26	0	0	0	0	6.6	EUR 6.6M
Aaron Wanless	West Ham	DF	26	2	5	1.2	3.5	12.27072	EUR 12.3M
Abdoulaye Doucoure	Everton	MF	31	3	2	4.1	2.4	13.5135	EUR 13.5M
Abdulkodir Abdullayev	Manchester City	DF	20	0	0	0	0.1	12.012	EUR 12.0M
Abdul Fatawu	Leicester City	FW	20	0	2	0.4	1.6	23.625	EUR 23.6M
Adam Armstrong	Southampton	FW, MF	27	2	2	3.3	1.2	21.57525	EUR 21.6M
Adam Lallana	Southampton	MF	36	0	2	0.2	0.9	10.6182	EUR 10.6M
Adam Smith	Bournemouth	DF	33	0	0	0.7	0.4	8.97768	EUR 9.0M
Adam Webster	Brighton	DF	29	0	0	0.2	0.5	9.90888	EUR 9.9M
Adam Wharton	Crystal Palace	MF	20	0	2	0.4	3	15.96	EUR 16.0M
Adama Traore	Fulham	FW, MF	28	2	7	4	4.9	23.3226	EUR 23.3M
Alber El Zoubi	Southampton	FW, MF	23	0	0	0.1	0	21.17115	EUR 21.2M
Alejandro Manches	Manchester City	FW	20	6	2	7.3	4.5	28.53	EUR 28.5M
Alex Iwobi	Fulham	FW, MF	28	9	6	4.7	7	26.2143	EUR 26.2M
Alex McCosker	Southampton	GK	34	0	0	0	0	5.4	EUR 5.4M
Alex Palmer	Ipswich Town	GK	27	0	0	0	0	6.45	EUR 6.5M
Alex Scott	Bournemouth	MF	20	0	0	0.7	0.9	15.24	EUR 15.2M
Alexander Isak	Newcastle	FW	24	23	6	20.3	4.3	37.1772	EUR 37.2M
Alexis Mac Allister	Liverpool	MF	25	5	5	2.8	4.7	16.875	EUR 16.9M
Ali Al-Hamrani	Ipswich Town	FW	22	0	0	0.4	0.1	21.708	EUR 21.7M
Alisson Becker	Liverpool	GK	31	0	0	0	0.6	5.8851	EUR 5.9M
Alphonse Areola	West Ham	GK	31	0	0	0	0	5.85	EUR 5.8M
Altay Bayraktar	Manchester City	GK	26	0	0	0	0	6.6	EUR 6.6M
Amad Diallo	Manchester City	FW, MF	22	8	6	4.7	4.9	29.0736	EUR 29.1M
Amadou Keita	Aston Villa	MF	22	3	0	2.1	0.7	15.6672	EUR 15.7M
Andreas Pereira	Fulham	MF	28	2	4	3.8	4.6	14.9184	EUR 14.9M
Andrew Robertson	Liverpool	DF	30	0	1	1.2	4.6	10.3008	EUR 10.3M
Andriy Shevchenko	Wolves	MF	23	0	0	0.4	0.8	14.2692	EUR 14.3M
Andriy Yarmolenko	Manchester City	GK	28	0	0	0	0.2	6.3126	EUR 6.3M
Andriy Zinchenko	Aston Villa	DF, MF	21	0	0	0	0.2	14.7294	EUR 14.7M
Andy Irwin	West Ham	MF	24	0	0	0.1	0.2	13.8414	EUR 13.8M
Anthony Elanga	Manchester City	FW, MF	22	6	11	4.5	5.7	29.9592	EUR 30.0M
Anthony Elanga	Newcastle	FW, MF	23	6	5	8	5	28.02375	EUR 28.0M
Antoine Semenyo	Bournemouth	FW	24	11	5	10	5.8	30.0771	EUR 30.1M

IV – Phân cụm cầu thủ bằng K-means và PCA

1. Lý giải cách làm

Ở câu IV, yêu cầu là:

- Sử dụng thuật toán K-means để phân loại các cầu thủ thành những nhóm có chỉ số tương tự nhau.

- Dùng các biểu đồ Elbow và Silhouette để lựa chọn số cụm K hợp lý, đưa ra nhận xét.

- Dùng thuật toán PCA để giảm số chiều dữ liệu xuống 2 chiều (và có thể mở

rộng lên 3 chiều), vẽ scatter plot thể hiện các cụm trên mặt phẳng 2D (và khối 3D).

Code nhóm viết triển khai các ý trên theo các bước chính sau:

- Đầu tiên, chương trình dùng `os.path` xác định đường dẫn tới file `results.csv` (chứa toàn bộ chỉ số câu thủ thu được từ phần I), sau đó đọc file bằng `pandas.read_csv` trong khối `try/except` để xử lý các lỗi như: file không tồn tại, file rỗng hoặc lỗi parser. Nếu đọc thành công, `DataFrame` `df` chứa đầy đủ dữ liệu ban đầu.

- Vì K-means chỉ làm việc với dữ liệu số, chương trình dùng `df.select_dtypes(include=[np.number])` để lấy ra `DataFrame` `df_numeric` chỉ gồm các cột kiểu số. Sau đó:

- In ra số lượng cột số ban đầu để kiểm tra.
- Xóa các cột hoàn toàn trống (toàn NaN) bằng `df_numeric.dropna(axis=1, how='all')`. Nếu có cột nào bị loại, chương trình in cảnh báo.

- Xử lý missing values: nếu `df_numeric` vẫn còn giá trị NaN, chương trình thống kê tổng số missing values rồi lần lượt duyệt qua từng cột. Với mỗi cột có NaN, median của cột được tính, nếu median cũng bị NaN (toàn bộ cột trống) thì gán 0; ngược lại dùng median đó để fillna. Cách làm này tương đương với chiến lược "imputation bằng trung vị", giúp dữ liệu đầy đủ mà không bị ảnh hưởng quá nhiều bởi ngoại lệ.

- Chương trình kiểm tra lại kích thước `df_numeric` sau khi xử lý. Nếu không còn cột số nào thì dừng lại. Ngoài ra, nếu xuất hiện giá trị vô cực (`inf`, `-inf`) thì thay bằng 0 để tránh lỗi khi chuẩn hóa.

- Trước khi phân cụm, dữ liệu được chuẩn hóa bằng `StandardScaler`. Câu lệnh `scaler.fit_transform(df_numeric)` trả về ma trận `X_scaled` có trung bình 0 và phương sai 1 cho từng cột. Chuẩn hóa rất quan trọng vì K-means dựa trên khoảng cách Euclid: nếu các cột có thang đo khác nhau thì cột lớn hơn sẽ chi phối hoàn toàn kết quả phân cụm.

• Để chọn số cụm K hợp lý, chương trình dùng hai tiêu chí:

1) Elbow method:

- Duyệt K từ 2 đến 10. Với mỗi K, khởi tạo mô hình `KMeans(n_clusters=k, random_state=42, n_init='auto')`, fit trên `X_scaled` và lấy `inertia` (tổng bình phương khoảng cách từ mỗi điểm đến tâm cụm) lưu vào danh sách `inertia`.

- Vẽ biểu đồ đường `inertia` theo K. Khi K tăng, `inertia` giảm dần; điểm

"khủy" (elbow) là vị trí inertia bắt đầu giảm chậm lại, gợi ý số cụm hợp lý.

2) Silhouette score:

- Với cùng vòng lặp K từ 2 đến 10, sau khi fit KMeans, chương trình tính `silhouette_score(X_scaled, kmeans.labels_)` và lưu vào `silhouette_scores`.

- Vẽ biểu đồ silhouette score theo K. Giá trị silhouette càng cao chứng tỏ các điểm trong cùng cụm càng gần nhau và các cụm phân tách rõ ràng. K cho score cao nhất (hoặc gần cao nhất mà vẫn phù hợp với Elbow) sẽ được chọn.

- Hai biểu đồ Elbow và Silhouette được vẽ trên cùng một figure (2 subplot) và lưu xuống file PNG với tên `elbow_silhouette_<timestamp>.png`. Đây là hai biểu đồ mà đề bài gợi ý.

- Sau khi quan sát hai biểu đồ, em chọn K tối ưu là 4 (gán vào biến `optimal_k = 4`). Lý do là:

- Ở K=4, đường Elbow bắt đầu gãy rõ (giảm chậm hơn khi tăng K).
- Silhouette score đạt giá trị tương đối cao so với các K khác (không bị giảm mạnh khi tăng số cụm).

Việc cố định K=4 trong code thể hiện quyết định này.

- Tiếp theo, chương trình khởi tạo lại KMeans với `n_clusters=optimal_k`, fit trên `X_scaled` và lấy nhãn cụm bằng `kmeans.fit_predict(X_scaled)`. Nhãn cụm (0,1,2,3) được thêm vào DataFrame gốc df dưới cột 'Cluster'.

- Để trực quan hóa và giảm chiều, chương trình dùng PCA với `n_components=2`. Dữ liệu chuẩn hóa `X_scaled` được biến đổi sang không gian 2 thành phần chính `X_pca`. Hai trục PCA này là những hướng phương sai lớn nhất trong dữ liệu, giúp biểu diễn cấu trúc phân cụm trên mặt phẳng 2D.

- Em định nghĩa một dict `cluster_names` để gán ý nghĩa cho từng cụm (dựa trên việc quan sát các chỉ số trong từng cụm):

- Cụm 0: "Hậu vệ/Phòng ngự" – nhóm cầu thủ có chỉ số phòng ngự nổi bật.
- Cụm 1: "Tiền vệ trung tâm" – nhóm cầu thủ cân bằng giữa phòng ngự và phân phối bóng.
- Cụm 2: "Tiền đạo/Tấn công" – nhóm có số bàn thắng, xG, xAG cao.
- Cụm 3: "Tiền vệ cánh/Công" – nhóm chơi rộng, hỗ trợ tấn công và chuyền bóng.

Tên nhóm được gán vào cột 'Tên Nhóm' của df dựa trên giá trị 'Cluster'.

- Cuối cùng, chương trình vẽ scatter plot 2D: trục hoành là thành phần chính 1, trục tung là thành phần chính 2, mỗi điểm là một cầu thủ, màu sắc biểu diễn nhóm cụm thông qua cột 'Tên Nhóm'. Biểu đồ được vẽ bằng `seaborn.scatterplot` và lưu xuống file 'bieu_do_phan_cum_co_chu_thich.png'. Biểu đồ này cho phép

quan sát trực quan cách các cụm được K-means tách ra trong không gian PCA 2 chiều.

Nếu muốn mở rộng đúng theo đề, có thể thêm một bước PCA với `n_components=3` và vẽ scatter plot 3D (dùng matplotlib Axes3D) để minh họa các cụm trong không gian 3 chiều; tuy nhiên code hiện tại đã thực hiện đầy đủ phân phân cụm, chọn K bằng Elbow + Silhouette và trực quan hóa 2D.

2. Thuật toán áp dụng

Bước 1: Đọc và chuẩn bị dữ liệu

1. Xác định đường dẫn tới `results.csv` và đọc dữ liệu vào DataFrame `df` bằng `pandas.read_csv` (kèm xử lý lỗi file).
2. Lấy các cột số bằng `df.select_dtypes(include=[np.number])` → `df_numeric`.
3. Loại bỏ các cột hoàn toàn trống: `df_numeric.dropna(axis=1, how='all')`.
4. Nếu còn NaN trong `df_numeric`:
 - Thống kê số lượng missing values.
 - Với mỗi cột có NaN, tính median của cột.
 - Nếu median là NaN (cột gần như trống), thay bằng 0; ngược lại dùng median đó để fillna cho cột.
5. Nếu trong `df_numeric` còn giá trị vô cực (`np.inf`, `-np.inf`), thay tất cả bằng 0.

Bước 2: Chuẩn hóa dữ liệu

6. Khởi tạo `StandardScaler` và gọi `scaler.fit_transform(df_numeric)` để thu được ma trận `X_scaled` có trung bình 0 và phương sai 1 cho từng thuộc tính.

Bước 3: Tìm số cụm K bằng Elbow và Silhouette

7. Khởi tạo hai list `inertia` và `silhouette_scores`; đặt dải `K_range = range(2, 11)`.
8. Với mỗi `k` trong `K_range`:
 - Khởi tạo `KMeans(n_clusters=k, random_state=42, n_init='auto')` và fit trên `X_scaled`.
 - Ghi lại `inertia_k = kmeans.inertia_` vào list `inertia`.
 - Tính `sil_k = silhouette_score(X_scaled, kmeans.labels_)` và lưu vào `silhouette_scores`.
9. Vẽ hai biểu đồ trên cùng một figure:

- Biểu đồ Elbow: trục x là K, trục y là inertia.
- Biểu đồ Silhouette: trục x là K, trục y là silhouette_scores.

Cả hai biểu đồ được lưu xuống file elbow_silhouette_<timestamp>.png.

10. Quan sát hai biểu đồ, chọn K tối ưu (trong code đặt optimal_k = 4) dựa trên điểm gãy Elbow và giá trị Silhouette cao.

Bước 4: Phân cụm K-means với K tối ưu

11. Khởi tạo lại KMeans(n_clusters=optimal_k, random_state=42, n_init='auto').

12. Gọi clusters = kmeans.fit_predict(X_scaled) để nhận nhãn cụm cho từng cầu thủ.

13. Thêm cột 'Cluster' vào DataFrame gốc df bằng df['Cluster'] = clusters.

Bước 5: Giảm chiều dữ liệu bằng PCA và vẽ biểu đồ phân cụm 2D

14. Khởi tạo PCA(n_components=2) và gọi X_pca = pca.fit_transform(X_scaled) để thu được tọa độ mới của từng cầu thủ trong không gian 2 thành phần chính.

15. Định nghĩa dict cluster_names ánh xạ nhãn cụm (0,1,2,3) sang tên nhóm vị trí (ví dụ: 'Hậu vệ/Phòng ngự', 'Tiền vệ trung tâm', 'Tiền đạo/Tấn công', 'Tiền vệ cánh/Công').

16. Thêm cột 'Tên Nhóm' vào df bằng cách map cột 'Cluster' qua cluster_names.

17. Vẽ scatter plot 2D với seaborn.scatterplot:

- Trục x: X_pca[:, 0] (thành phần chính 1).
- Trục y: X_pca[:, 1] (thành phần chính 2).
- Màu sắc (hue): df['Tên Nhóm'] để hiển thị các cụm khác nhau.
- Đặt tiêu đề, nhãn trục và legend, sau đó lưu hình ra file

'bieu_do_phan_cum_co_chu_thich.png'.

Kết quả của thuật toán là:

- Các cầu thủ được phân vào 4 cụm dựa trên bộ chỉ số đã chuẩn hóa.
- Hai biểu đồ Elbow và Silhouette giúp giải thích rõ lý do chọn K=4.
- Biểu đồ PCA 2D cho thấy cấu trúc các cụm trong không gian giảm chiều, giúp trực quan hóa các nhóm cầu thủ có phong cách/đặc trưng tương tự nhau.

3. Kết quả chạy được

