HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

BÁO CÁO BÀI TẬP LỚN I

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn	**: Kim Ngọc Bách**
Sinh viên	**: Nguyễn Duy Thắng**
Mã sinh viên	**: B23DCCE087**
Lớp	**: D23CQCE06-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 2025

Giảng viên

MỤC LỤC

Lời mở đầu

Bóng đá, với sức hút toàn cầu, không chỉ là cuộc chiến trên sân cỏ mà còn là một lĩnh vực mà khoa học dữ liệu đang tạo ra những bước tiến vượt bậc. Giải Ngoại hạng Anh (Premier League) mùa giải 2024-2025, nơi quy tụ các đội bóng hàng đầu như Manchester City, Liverpool, và Arsenal, là một kho dữ liệu khổng lồ về hiệu suất thi đấu của cầu thủ. Từ số bàn thắng, kiến tạo, đến các chỉ số chuyên sâu như bàn thắng kỳ vọng (xG), khoảng cách chuyền bóng tiến bộ (PrgP), hay tỷ lệ cứu thua, mỗi trận đấu tạo ra hàng triệu điểm dữ liệu, cung cấp cái nhìn sâu sắc về năng lực cá nhân và tập thể. Các nguồn dữ liệu như FBref, với thống kê chi tiết về mọi khía cạnh của trận đấu, và Football Transfers, với thông tin về giá trị thị trường, đã trở thành công cụ không thể thiếu cho các câu lạc bộ, huấn luyện viên, và nhà phân tích. Trong bối cảnh này, lập trình Python, với các thư viện mạnh mẽ như `pandas`, `matplotlib`, và `scikit-learn`, đã mở ra cơ hội khai thác dữ liệu bóng đá một cách hiệu quả, biến những con số thô thành thông tin chiến lược.

Xu hướng ứng dụng khoa học dữ liệu trong bóng đá, lấy cảm hứng từ các phương pháp như Moneyball, đang định hình lại cách các đội bóng vận hành. Các chỉ số như xG giúp đánh giá hiệu quả tấn công, trong khi PrgP và PrgC phản ánh đóng góp chiến thuật của cầu thủ. Các câu lạc bộ lớn ngày càng dựa vào phân tích dữ liệu để phát hiện tài năng trẻ, tối ưu hóa đội hình, và cạnh tranh trong thị trường chuyển nhượng, nơi giá trị cầu thủ có thể lên đến hàng trăm triệu euro. Bài tập lập trình Python này được thiết kế để đáp ứng nhu cầu thực tiễn và học thuật, khai thác dữ liệu thống kê cầu thủ từ FBref cho mùa giải Premier League 2024-2025, tập trung vào các cầu thủ thi đấu trên 90 phút, và giá trị chuyển nhượng cho những ai chơi trên 900 phút. Các nhiệm vụ chính bao gồm phân tích đề bài để hiểu rõ yêu cầu, thu thập và xử lý dữ liệu, phân tích thống kê, phân cụm cầu thủ, và ước lượng giá trị chuyển nhượng, tất cả được thực hiện bằng Python và các công cụ như `worldfootballR`.

Cụ thể, báo cáo sẽ bắt đầu bằng việc phân tích đề bài, làm rõ mục tiêu và yêu cầu. Tiếp theo, phần thuật toán và phân tích thuật toán sẽ trình bày chi tiết quá trình thu thập dữ liệu từ FBref, xử lý và lưu trữ vào file `results.csv`, phân tích thống kê để xác định top 3 cầu thủ cao nhất/thấp nhất, tính toán các chỉ số mô tả (median, mean, standard deviation), và vẽ histogram phân phối. Phần này cũng bao gồm phân cụm cầu thủ bằng thuật toán K-means, giảm chiều dữ liệu bằng PCA, và ước lượng giá trị chuyển nhượng từ Football Transfers bằng mô hình học máy. Phần kết luận và thảo luận sẽ tóm tắt các phát hiện, nêu khó khăn gặp phải, và đề xuất cải tiến. Tài liệu tham khảo và phụ lục cung cấp mã nguồn, biểu đồ, và dữ liệu bổ sung để minh họa phân tích.

Phạm vi của báo cáo giới hạn ở mùa giải Premier League 2024-2025, với dữ liệu từ FBref và Football Transfers. Kết quả phân tích mang lại giá trị thực tiễn, hỗ trợ các câu lạc bộ đánh giá hiệu suất, xác định đội bóng xuất sắc nhất, và đưa ra quyết định chuyển nhượng chính xác. Đồng thời, bài tập là cơ hội thực hành các kỹ năng khoa học dữ liệu, từ thu thập dữ liệu đến học máy, thể hiện sự giao thoa giữa công nghệ và thể thao. Báo cáo được tổ chức theo các phần trong mục lục, bắt đầu bằng phân tích đề bài và kết thúc bằng kết luận, đảm bảo tính logic và toàn diện. Phần tiếp theo sẽ trình bày chi tiết quá trình phân tích đề bài, đặt nền tảng cho các phân tích chuyên sâu.

I. Phân tích đề bài

1. Mục tiêu của bài tập

Bài tập lập trình Python này nhằm khai thác và phân tích dữ liệu thống kê cầu thủ từ mùa giải Premier League 2024-2025, một trong những giải đấu bóng đá hàng đầu thế giới, để đánh giá hiệu suất cá nhân, xác định đội bóng xuất sắc, và ước lượng giá trị chuyển nhượng. Mục tiêu tổng quát là áp dụng các kỹ thuật khoa học dữ liệu, bao gồm thu thập, xử lý, phân tích, và trực quan hóa dữ liệu, để cung cấp thông tin có giá trị cho các câu lạc bộ, huấn luyện viên, và nhà phân tích thể thao. Cụ thể, bài tập yêu cầu thực hiện các nhiệm vụ sau: thu thập dữ liệu thống kê từ FBref cho các cầu thủ thi đấu trên 90 phút và lưu vào file `results.csv`; phân tích thống kê để xác định top 3 cầu thủ cao nhất và thấp nhất, tính toán các chỉ số mô tả như median, mean, và độ lệch chuẩn, đồng thời vẽ biểu đồ histogram để phân tích phân phối; phân cụm cầu thủ bằng thuật toán K-means và trực quan hóa bằng PCA; và cuối cùng, thu thập dữ liệu giá trị chuyển nhượng từ Football Transfers cho các cầu thủ chơi trên 900 phút, đồng thời đề xuất phương pháp ước lượng giá trị bằng mô hình học máy.

Mục tiêu của bài tập không chỉ mang tính thực tiễn, hỗ trợ các câu lạc bộ trong việc đánh giá hiệu suất và đưa ra quyết định chuyển nhượng, mà còn có giá trị học thuật, cung cấp cơ hội thực hành các kỹ năng khoa học dữ liệu như web scraping, phân tích thống kê, và học máy. Các nhiệm vụ này sẽ được triển khai chi tiết trong các phần tiếp theo của báo cáo, từ thu thập dữ liệu, phân tích thuật toán, đến kết luận và đề xuất cải tiến, nhằm đảm bảo một quy trình phân tích toàn diện và logic.

2. Tổng quan về dữ liệu và yêu cầu

Dữ liệu được sử dụng trong bài tập đến từ hai nguồn chính: FBref và Football Transfers. FBref cung cấp dữ liệu thống kê chi tiết về hiệu suất cầu thủ trong mùa giải Premier League 2024-2025, bao gồm các danh mục như hiệu suất (bàn thắng, kiến tạo, thẻ phạt), chuyền bóng (tỷ lệ hoàn thành, khoảng cách chuyền), phòng ngự (tắc bóng, chặn bóng), thủ môn (tỷ lệ cứu thua, bàn thua), và nhiều chỉ số khác như xG, PrgP, và SCA. Dữ liệu này chỉ được thu thập cho các cầu thủ thi đấu trên 90 phút, đảm bảo tập trung vào những người có đóng góp đáng kể. Trong khi đó, Football Transfers cung cấp thông tin về giá trị chuyển nhượng, nhưng chỉ áp dụng cho các cầu thủ chơi trên 900 phút, phản ánh giá trị thị trường của những cầu thủ chủ chốt.

Các yêu cầu kỹ thuật của bài tập bao gồm: thu thập dữ liệu từ FBref và lưu trữ vào file `results.csv` với các cột tương ứng với các chỉ số thống kê, sắp xếp cầu thủ theo thứ tự alphabet tên; phân tích thống kê để xác định top 3 cầu thủ cao nhất và thấp nhất cho mỗi chỉ số, lưu vào file `top_3.txt`, và tính toán median, mean, độ lệch chuẩn cho tất cả cầu thủ và từng đội, lưu vào `results2.csv`; vẽ biểu đồ histogram để phân tích phân phối các chỉ số; xác định đội bóng có điểm số cao nhất cho mỗi chỉ số và đánh giá đội xuất sắc nhất; phân cụm cầu thủ bằng thuật toán K-means, giảm chiều dữ liệu bằng PCA, và trực quan hóa kết quả; cuối cùng, thu thập giá trị chuyển nhượng từ Football Transfers và đề xuất phương pháp ước lượng giá trị bằng mô hình học máy, với việc lựa chọn đặc trưng và đánh giá mô hình.

Để thực hiện các nhiệm vụ này, bài tập sử dụng Python làm công cụ chính, kết hợp với các thư viện như `pandas` để xử lý dữ liệu, `matplotlib` để trực quan hóa, `scikit-learn` để phân cụm và học máy, và `worldfootballR` để thu thập dữ liệu từ FBref. Các kỹ thuật được áp dụng bao gồm web scraping, phân tích thống kê, phân cụm, và

học máy, đảm bảo một quy trình phân tích toàn diện. Các yêu cầu này sẽ được triển khai chi tiết trong Phần II (Thuật toán và phân tích thuật toán), bắt đầu bằng quá trình thu thập và xử lý dữ liệu từ FBref.

II. Thuật toán và phân tích thuật toán

1. Thu thập và xử lý dữ liệu thống kê cầu thủ

1.1. Mô tả nguồn dữ liệu từ FBref

Trang web FBref là nguồn dữ liệu chính, cung cấp thống kê chi tiết về hiệu suất cầu thủ trong mùa giải Premier League 2024-2025. Dữ liệu bao gồm các danh mục như:

- **Hiệu suất**: bàn thắng (Gls), kiến tạo (Ast), thẻ vàng (crdY), thẻ đỏ (crdR).
- **Chuyền bóng**: số đường chuyền hoàn thành (Cmp), tỷ lệ hoàn thành (Cmp%), khoảng cách chuyền (TotDist).
- **Phòng ngự**: số lần tắc bóng (Tkl), chặn bóng (Blocks), đánh chặn (Int).
- **Thủ môn**: tỷ lệ cứu thua (Save%), bàn thua mỗi 90 phút (GA90), tỷ lệ giữ sạch lưới (CS%).
- **Chỉ số tiến bộ**: bàn thắng kỳ vọng (xG), kiến tạo kỳ vọng (xAG), chuyền bóng tiến bộ (PrgP), dẫn bóng tiến bộ (PrgC), nhận bóng tiến bộ (PrgR).
- **Khác**: hành động tạo cơ hội (SCA, GCA), tranh chấp trên không (Aerl Won, Aerl Lost).

Dữ liệu được tổ chức trong các bảng HTML, với các định danh như ***stats_standard, stats_keeper, stats_shooting,*** trên các trang như ***/stats, /keepers, /shooting, /passing,*** v.v. Các bảng này nằm trong phần bình luận HTML, yêu cầu kỹ thuật đặc biệt để truy xuất. Phạm vi thu thập giới hạn ở các cầu thủ thi đấu trên 90 phút, đảm bảo tập trung vào những người có đóng góp đáng kể trong mùa giải.

1.2. Phương pháp thu thập dữ liệu

Hàm setup_driver

Mã:

```
def setup_driver():
    options = Options()
    options.add_argument("--headless")
    driver = webdriver.Chrome(options=options)
    return driver
```

Cách thức hoạt động:

1. **Tạo đối tượng cấu hình**:
    - Dòng options = Options() khởi tạo một đối tượng Options từ thư viện selenium.webdriver.chrome.options. Đối tượng này cho phép tùy chỉnh cách trình duyệt Chrome hoạt động.
    - Ví dụ: Có thể thêm các tham số như chế độ ẩn, kích thước cửa sổ, hoặc vô hiệu hóa tiện ích mở rộng.
2. **Cấu hình chế độ headless**:
    - options.add_argument("--headless") thêm tham số "--headless" vào cấu hình, yêu cầu Chrome chạy mà không hiển thị giao diện đồ họa (GUI). Điều này:

        - Tiết kiệm tài nguyên CPU và RAM, đặc biệt khi chạy trên máy chủ hoặc xử lý hàng loạt.
        - Tăng tốc độ tải trang vì không cần render giao diện.
    - Ví dụ: Thay vì mở Chrome và hiển thị trang FBref, trình duyệt xử lý ngầm và chỉ trả về mã nguồn HTML.
3. **Khởi tạo trình duyệt**:
    - driver = webdriver.Chrome(options=options) tạo một đối tượng webdriver.Chrome, sử dụng ChromeDriver (phần mềm trung gian) để kết nối Python với trình duyệt Chrome.
    - ChromeDriver phải được cài đặt trước và khớp với phiên bản Chrome trên máy. Đối tượng driver này hoạt động như một trình duyệt ảo, có thể tải trang, thực thi JavaScript, và lấy mã nguồn.
4. **Trả về đối tượng driver**:
    - Hàm trả về driver, được sử dụng trong hàm scrape_table để truy cập các URL của FBref và tải nội dung động.
5. **Tương tác với quy trình**:
    - Trong hàm main, setup_driver được gọi một lần để tạo driver, sau đó driver được tái sử dụng cho tất cả URL trong stat_urls để thu thập bảng.
    - Sau khi hoàn thành, driver.quit() trong main đóng trình duyệt, giải phóng tài nguyên.

Ví dụ cụ thể:

- Khi chạy, setup_driver tạo một trình duyệt Chrome ẩn. Nếu truy cập trang https://fbref.com/en/comps/9/2024-2025/stats/, trình duyệt tải trang mà không hiển thị, sẵn sàng để scrape_table lấy mã nguồn.

Vai trò: Cung cấp công cụ để tải các trang web động của FBref, nơi bảng dữ liệu (như stats_standard) được tạo bởi JavaScript và ẩn trong bình luận HTML.

Xử lý lỗi:

- Nếu ChromeDriver không được cài đặt hoặc phiên bản không khớp, mã sẽ báo lỗi WebDriverException. Mã gốc có thể dùng try-except để thử lại hoặc thông báo lỗi.
- Lỗi cấu hình (như thiếu tham số) được giảm thiểu bằng cách sử dụng Options mặc định.

Hàm scrape_table

Mã:

```
def scrape_table(driver, url, table_id):
    driver.get(url)
    time.sleep(3)
    page_soup = BeautifulSoup(driver.page_source, "html.parser")
    for comment in page_soup.find_all(string=lambda text: isinstance(text, Comment)):
        if table_id in comment:
            comment_soup = BeautifulSoup(comment, "html.parser")
            table = comment_soup.find("table", {"id": table_id})
            if table:
                return pd.read_html(StringIO(str(table)))[0]
    return None
```

Cách thức hoạt động:

1. **Tải trang web**:
   - driver.get(url) yêu cầu trình duyệt tải trang tại url (ví dụ: https://fbref.com/en/comps/9/2024-2025/stats/2024-2025-Premier-League-Stats).
   - Lệnh này gửi yêu cầu HTTP đến FBref, tải toàn bộ nội dung trang, bao gồm HTML tĩnh và các phần động do JavaScript tạo ra (như bảng stats_standard).
   - time.sleep(3) tạm dừng 3 giây để đảm bảo trang tải hoàn tất. Điều này cần thiết vì:
     - FBref sử dụng JavaScript để render bảng, có thể mất vài giây để hoàn thành.
     - Nếu lấy mã nguồn ngay lập tức, bảng có thể chưa xuất hiện, dẫn đến lỗi.
2. **Phân tích mã nguồn HTML**:
   - driver.page_source lấy toàn bộ mã nguồn HTML của trang sau khi JavaScript thực thi.
   - page_soup = BeautifulSoup(driver.page_source, "html.parser") chuyển mã nguồn thành đối tượng BeautifulSoup, cho phép phân tích cú pháp HTML (tìm thẻ, thuộc tính, v.v.).
3. **Tìm bình luận HTML chứa bảng**:
   - page_soup.find_all(string=lambda text: isinstance(text, Comment)) tìm tất cả các đoạn bình luận HTML () trong trang.
   - Lambda string=lambda text: isinstance(text, Comment) kiểm tra từng phần tử trong HTML, chỉ lấy những phần tử là bình luận (loại bỏ thẻ, văn bản thông thường).
   - FBref lưu trữ bảng thống kê (như stats_standard) trong bình luận HTML để ẩn khỏi giao diện mặc định, nên bước này là cần thiết.
4. **Tìm bảng cụ thể**:
   - Vòng lặp for comment in ... kiểm tra từng bình luận:
     - if table_id in comment kiểm tra xem table_id (ví dụ: "stats_standard") có trong nội dung bình luận không.
     - Nếu có, comment_soup = BeautifulSoup(comment, "html.parser") tạo một đối tượng BeautifulSoup riêng để phân tích bình luận đó.
     - table = comment_soup.find("table", {"id": table_id}) tìm thẻ <table> với thuộc tính id khớp (ví dụ: <table id="stats_standard">).
5. **Chuyển bảng thành DataFrame**:
   - Nếu bảng được tìm thấy (if table):
     - str(table) chuyển bảng HTML thành chuỗi.
     - StringIO(str(table)) tạo một đối tượng chuỗi để pandas.read_html đọc.
     - pd.read_html(...)[0] chuyển bảng HTML thành danh sách DataFrame, lấy DataFrame đầu tiên ([0]).
   - Ví dụ: Bảng stats_standard có các cột như Player, Team, Gls, Ast, được chuyển thành DataFrame với các hàng tương ứng từng cầu thủ.
6. **Trả về kết quả**:
   - Nếu tìm thấy bảng, trả về DataFrame.
   - Nếu không (bảng không tồn tại hoặc lỗi), trả về None.
7. **Tương tác với quy trình**:
   - Trong main, hàm được gọi cho từng cặp url và table_id trong stat_urls và table_identifiers.
   - Kết quả được truyền vào process_table để chuẩn hóa trước khi gộp.

Ví dụ cụ thể:

- Với url = "https://fbref.com/.../stats/..." và table_id = "stats_standard":
  - Trang được tải, chờ 3 giây.

- Mã tìm bình luận chứa "stats_standard", trích xuất bảng HTML như:

```
<table id="stats_standard">
  <tr><th>Player</th><th>Team</th><th>Gls</th>...</tr>
  <tr><td>Lionel Messi</td><td>Arsenal</td><td>10</td>...</tr>
  ...
</table>
```

- Bảng được chuyển thành DataFrame với các cột Player, Team, Gls, v.v.

Vai trò: Truy xuất bảng ẩn trong bình luận HTML, chuyển thành định dạng DataFrame để xử lý tiếp.

Xử lý lỗi:

- Thời gian chờ 3 giây giảm nguy cơ lỗi do tải trang chậm.
- Trả về None nếu bảng không tồn tại, cho phép main bỏ qua và tiếp tục với bảng khác.
- Try-except trong mã gốc xử lý lỗi mạng, HTML bất ngờ, hoặc định dạng bảng không hợp lệ.

1.3. Xử lý và lưu dữ liệu vào file results.csv

Hàm process_table

Mã:

```
def process_table(table_data, table_id, rename_map):
    table_data = table_data.rename(columns=rename_map.get(table_id, {}))
    table_data["Player"] = table_data["Player"].apply(lambda x: " ".join(x.split(",")[::-
1]).strip() if "," in x else x)
    table_data["Age"] = table_data["Age"].apply(lambda x: round(int(x.split("-")[0]) +
int(x.split("-")[1])/365, 2) if "-" in str(x) else x)
    return table_data
```

Cách thức hoạt động:

1. **Đổi tên cột**:
   - rename_map.get(table_id, {}) lấy ánh xạ tên cột từ rename_map (một từ điển định nghĩa trước, ví dụ: {"stats_standard": {"Goals": "Gls", "Assists": "Ast"}}).
   - Nếu table_id không có ánh xạ, trả về {} (từ điển rỗng), không đổi tên cột.
   - table_data.rename(columns=...) đổi tên cột theo ánh xạ. Ví dụ:
     - Trong bảng stats_standard, cột "Goals" đổi thành "Gls", "Assists" thành "Ast".
     - Điều này đảm bảo các bảng có tên cột thống nhất khi gộp.
2. **Chuẩn hóa tên cầu thủ**:
   - table_data["Player"].apply(...) áp dụng hàm xử lý cho từng giá trị trong cột Player.
   - Hàm lambda:
     - Nếu tên chứa dấu phẩy (ví dụ: "Messi, Lionel"), x.split(",") tách thành ["Messi", "Lionel"].
     - [::-1] đảo ngược thành ["Lionel", "Messi"].
     - " ".join(...) nối thành "Lionel Messi".

        - strip() loại bỏ khoảng trắng thừa.
        - Nếu không có dấu phẩy (ví dụ: "Son Heung-min"), giữ nguyên.
    - Ví dụ: "Messi, Lionel" thành "Lionel Messi", "Son Heung-min" giữ nguyên.
3. **Chuẩn hóa tuổi**:
    - table_data["Age"].apply(...) áp dụng hàm xử lý cho cột Age.
    - Hàm lambda:
        - Nếu tuổi dạng "years-days" (ví dụ: "25-123"), x.split("-") tách thành ["25", "123"].
        - int(x.split("-")[0]) + int(x.split("-")[1])/365 tính tuổi thập phân: 25 + 123/365 ≈ 25.34.
        - round(..., 2) làm tròn 2 chữ số thập phân.
        - Nếu không có dấu gạch ngang (ví dụ: giá trị thiếu hoặc định dạng khác), giữ nguyên.
    - Ví dụ: "25-123" thành 25.34, giá trị thiếu giữ nguyên.
4. **Trả về DataFrame**:
    - Trả về table_data với cột đã đổi tên, tên cầu thủ và tuổi chuẩn hóa.
5. **Tương tác với quy trình**:
    - Được gọi ngay sau scrape_table trong main, xử lý từng bảng trước khi lưu vào tables để gộp.

Ví dụ cụ thể:

- Với bảng stats_standard có cột [Player, Goals, Assists]:
    - Đổi "Goals" thành "Gls", "Assists" thành "Ast".
    - Chuẩn hóa "Messi, Lionel" thành "Lionel Messi".
    - Chuẩn hóa tuổi "25-123" thành 25.34.

Vai trò: Chuẩn hóa dữ liệu để đảm bảo định dạng thống nhất, dễ gộp và phân tích.

Xử lý lỗi:

- rename_map.get(table_id, {}) tránh lỗi nếu table_id không có ánh xạ.
- Kiểm tra "," in x và "-" in str(x) tránh lỗi định dạng tên hoặc tuổi.
- Giá trị thiếu được giữ nguyên, xử lý sau trong finalize_data.

Hàm merge_tables

Mã:

```
def merge_tables(tables, target_columns):
    final_data = None
    for table_id, table_data in tables.items():
        table_data = table_data[[col for col in table_data.columns if col in target_columns]]
        if final_data is None:
            final_data = table_data
        else:
            final_data = pd.merge(final_data, table_data, on="Player", how="outer")
    return final_data
```

Cách thức hoạt động:

1. **Khởi tạo DataFrame tổng hợp**:

- final_data = None khởi tạo biến để chứa dữ liệu gộp từ tất cả bảng.

2. **Lặp qua các bảng**:
   - tables.items() trả về cặp table_id (ví dụ: stats_standard) và table_data (DataFrame từ process_table).
3. **Lọc cột cần thiết**:
   - [col for col in table_data.columns if col in target_columns] tạo danh sách các cột có trong target_columns (danh sách cột mong muốn, như Player, Team, Gls, xG).
   - table_data[...] chỉ giữ các cột này, loại bỏ cột dư thừa (ví dụ: cột tạm thời hoặc không cần thiết).
   - Ví dụ: Bảng stats_standard có cột [Player, Team, Gls, Ast, Temp], chỉ giữ [Player, Team, Gls, Ast].
4. **Gộp bảng**:
   - Nếu final_data là None (bảng đầu tiên), gán final_data = table_data.
   - Nếu không, pd.merge(final_data, table_data, on="Player", how="outer"):
     - Gộp final_data với table_data dựa trên cột Player (khóa chính).
     - how="outer" giữ tất cả cầu thủ, ngay cả khi họ chỉ xuất hiện trong một bảng.
     - Giá trị thiếu (cho cột không có dữ liệu) được điền bằng NaN.
   - Ví dụ: Gộp bảng stats_standard (Player, Gls) với stats_shooting (Player, xG):
     - Kết quả có cột [Player, Gls, xG].
     - Cầu thủ chỉ có trong stats_shooting (như thủ môn) sẽ có NaN ở cột Gls.
5. **Trả về DataFrame**:
   - Trả về final_data chứa tất cả cột từ các bảng, với dữ liệu gộp.
6. **Tương tác với quy trình**:
   - Được gọi trong main sau khi xử lý tất cả bảng, tạo DataFrame tổng hợp để truyền vào finalize_data.

Ví dụ cụ thể:

- Với tables chứa:
  - stats_standard: [Player, Team, Gls] (Lionel Messi, Arsenal, 10).
  - stats_shooting: [Player, xG] (Lionel Messi, 8.5).
- Sau gộp, final_data có cột [Player, Team, Gls, xG], hàng: [Lionel Messi, Arsenal, 10, 8.5].

Vai trò: Tạo DataFrame tổng hợp từ nhiều bảng, đảm bảo đầy đủ thông tin từ các danh mục (hiệu suất, chuyền bóng, phòng ngự, v.v.).

Xử lý lỗi:

- target_columns đảm bảo chỉ giữ cột hợp lệ.
- how="outer" xử lý trường hợp cầu thủ chỉ có trong một bảng.
- Mã gốc kiểm tra table_data không phải None trước khi gộp.

Hàm finalize_data

```
def finalize_data(data, target_columns):
    data = data[data["Minutes"] > 90]
    data["Gls"] = pd.to_numeric(data["Gls"], errors="coerce").astype("Int64")
    data["xG"] = pd.to_numeric(data["xG"], errors="coerce").round(2)
    data["Player"] = data["Player"].fillna("N/A")
    return data.sort_values("Player")
```

Cách thức hoạt động:

1. **Lọc cầu thủ**:
   - data[data["Minutes"] > 90] chỉ giữ các hàng có cột Minutes (thời gian thi đấu) lớn hơn 90 phút.
   - Điều này loại bỏ các cầu thủ ít ra sân, tập trung vào những người có đóng góp đáng kể.
   - Ví dụ: Một cầu thủ với Minutes = 50 bị loại; Minutes = 100 được giữ.
2. **Định dạng cột Gls**:
   - pd.to_numeric(data["Gls"], errors="coerce") chuyển cột Gls thành số:
     - Giá trị hợp lệ (như "10") thành 10.
     - Giá trị không hợp lệ (như chuỗi hoặc rỗng) thành NaN.
   - .astype("Int64") chuyển thành số nguyên, với kiểu Int64 hỗ trợ giá trị thiếu (NaN).
   - Ví dụ: "10" thành 10, "N/A" thành NaN.
3. **Định dạng cột xG**:
   - pd.to_numeric(data["xG"], errors="coerce") chuyển cột xG thành số, tương tự Gls.
   - .round(2) làm tròn 2 chữ số thập phân (ví dụ: 8.567 thành 8.57).
   - Ví dụ: "8.5" thành 8.50, "N/A" thành NaN.
4. **Xử lý giá trị thiếu ở Player**:
   - data["Player"].fillna("N/A") thay các giá trị NaN trong cột Player bằng "N/A".
   - Điều này đảm bảo cột Player không có ô trống, hỗ trợ tra cứu và phân tích.
5. **Sắp xếp**:
   - data.sort_values("Player") sắp xếp DataFrame theo cột Player (theo thứ tự chữ cái, ví dụ: "Aaron Ramsey" trước "Zinedine Zidane").
   - Điều này giúp file đầu ra dễ tra cứu và nhất quán.
6. **Trả về DataFrame**:
   - Trả về data đã được lọc, định dạng, và sắp xếp.
7. **Tương tác với quy trình**:
   - Được gọi sau merge_tables trong main, chuẩn bị dữ liệu cuối cùng để lưu vào save_data.

Ví dụ cụ thể:

- Với DataFrame có hàng:
  - [Player: "Lionel Messi", Team: "Arsenal", Minutes: 100, Gls: "10", xG: "8.5"]
  - [Player: "Unknown", Team: "Chelsea", Minutes: 50, Gls: "0", xG: "N/A"]
- Sau xử lý:
  - Loại hàng Minutes = 50.
  - Gls: 10 (số nguyên), xG: 8.50 (2 chữ số thập phân).
  - Sắp xếp: "Lionel Messi" đứng đầu nếu là tên đầu tiên theo thứ tự chữ cái.

Vai trò: Hoàn thiện dữ liệu bằng cách lọc cầu thủ đủ điều kiện, định dạng đúng kiểu dữ liệu, và sắp xếp để dễ sử dụng.

Xử lý lỗi:

- errors="coerce" xử lý giá trị không hợp lệ trong cột số.
- Int64 hỗ trợ NaN trong số nguyên.
- Kiểm tra cột Minutes tồn tại trong mã gốc.
- fillna("N/A") đảm bảo cột Player không có giá trị rỗng.

Hàm save_data

Mã:

```
def save_data(data, output_path):
    data.to_csv(output_path, index=False, encoding="utf-8-sig", na_rep="N/A")
    print(f"Saved {data.shape[0]} rows to {output_path}")
```

Cách thức hoạt động:

1. **Lưu file CSV**:
    - data.to_csv(output_path, index=False, encoding="utf-8-sig", na_rep="N/A") lưu DataFrame vào file tại output_path (ví dụ: "results.csv"):
        - index=False không lưu chỉ số hàng, giữ file gọn gàng (không có cột 0, 1, 2...).
        - encoding="utf-8-sig" dùng UTF-8 với BOM (Byte Order Mark), đảm bảo tương thích với Excel, hiển thị đúng ký tự đặc biệt (như tên cầu thủ có dấu).
        - na_rep="N/A" thay tất cả giá trị NaN trong file bằng chuỗi "N/A", giúp file dễ đọc.
    - Ví dụ: Một hàng [Player: "Lionel Messi", Gls: 10, xG: NaN] được lưu thành "Lionel Messi,10,N/A".
2. **In thông báo**:
    - data.shape[0] lấy số hàng của DataFrame (số cầu thủ).
    - print(f"Saved {data.shape[0]} rows to {output_path}") in thông báo, ví dụ: "Saved 500 rows to results.csv".
    - Điều này giúp người dùng xác nhận file đã được lưu và biết số lượng dữ liệu.
3. **Tương tác với quy trình**:
    - Được gọi cuối cùng trong main, sau khi finalize_data hoàn thiện dữ liệu.
    - Kết quả là file results.csv, đầu vào cho mục II.2.

Ví dụ cụ thể:

- Với DataFrame 500 hàng, cột [Player, Team, Gls, xG]:
    - File results.csv được tạo với 500 dòng, mỗi dòng là một cầu thủ.
    - Dòng ví dụ: "Lionel Messi,Arsenal,10,8.50".
    - In: "Saved 500 rows to results.csv".

Vai trò: Lưu dữ liệu đã xử lý vào file CSV, tạo đầu ra sạch và sẵn sàng cho phân tích.

Xử lý lỗi:

- Try-except trong mã gốc xử lý lỗi ghi file (như thư mục không tồn tại, quyền truy cập).
- UTF-8-sig tránh lỗi hiển thị ký tự đặc biệt trong Excel.

1.4. Kết quả và kiểm tra dữ liệu

Hàm main

Mã:

```
def main():
```

```
    stat_urls = ["https://fbref.com/en/comps/9/2024-2025/stats/2024-2025-Premier-League-
Stats", ...]
    table_identifiers = ["stats_standard", "stats_keeper", ...]
    driver = setup_driver()
    tables = {}
    for url, table_id in zip(stat_urls, table_identifiers):
        tables[table_id] = process_table(scrape_table(driver, url, table_id), table_id,
rename_columns_map)
    final_data = merge_tables(tables, target_columns)
    final_data = finalize_data(final_data, target_columns)
    save_data(final_data, "results.csv")
    driver.quit()
```

Cách thức hoạt động:

1. **Khởi tạo danh sách**:
   - stat_urls: Danh sách URL của FBref, mỗi URL tương ứng một danh mục (stats, keepers, shooting, passing, v.v.).
   - table_identifiers: Danh sách ID bảng (stats_standard, stats_keeper, stats_shooting, v.v.), khớp với URL.
   - Ví dụ: stat_urls[0] là trang stats, table_identifiers[0] là "stats_standard".
2. **Khởi tạo trình duyệt**:
   - driver = setup_driver() tạo trình duyệt Chrome headless, sẵn sàng tải trang.
3. **Thu thập và xử lý bảng**:
   - zip(stat_urls, table_identifiers) tạo các cặp URL và ID (ví dụ: (URL_stats, "stats_standard")).
   - Vòng lặp for url, table_id in ...:
     - scrape_table(driver, url, table_id) tải trang và trích xuất bảng, trả về DataFrame hoặc None.
     - process_table(...) chuẩn hóa bảng (đổi tên cột, tên cầu thủ, tuổi).
     - Kết quả lưu vào tables (từ điển với khóa là table_id, giá trị là DataFrame).
   - Ví dụ: tables["stats_standard"] chứa DataFrame với cột [Player, Team, Gls, Ast].
4. **Gộp bảng**:
   - merge_tables(tables, target_columns) gộp tất cả bảng trong tables thành một DataFrame duy nhất, giữ các cột trong target_columns.
5. **Hoàn thiện dữ liệu**:
   - finalize_data(final_data, target_columns) lọc cầu thủ (Minutes > 90), định dạng cột (Gls, xG), sắp xếp theo Player.
6. **Lưu file**:
   - save_data(final_data, "results.csv") lưu DataFrame vào results.csv.
7. **Đóng trình duyệt**:
   - driver.quit() đóng Chrome, giải phóng tài nguyên hệ thống.
8. **Tương tác với quy trình**:
   - Hàm main là điểm khởi đầu, điều phối toàn bộ quy trình từ thu thập đến lưu trữ.
   - Kết quả là file results.csv, sẵn sàng cho mục II.2.

Ví dụ cụ thể:

- Với 3 URL (stats, keepers, shooting) và 3 ID (stats_standard, stats_keeper, stats_shooting):
  - Tải từng trang, trích xuất bảng, chuẩn hóa, lưu vào tables.
  - Gộp thành final_data với cột [Player, Team, Gls, xG, Save%, ...].

- Lọc Minutes > 90, định dạng, lưu thành results.csv (~500 hàng).
- In: "Saved 500 rows to results.csv".

Vai trò: Kết nối tất cả hàm, chạy quy trình thu thập, xử lý, và lưu dữ liệu.

Xử lý lỗi:

- Kiểm tra scrape_table trả về None, bỏ qua bảng lỗi.
- Try-except trong mã gốc xử lý lỗi mạng, HTML, hoặc ghi file.
- driver.quit() đảm bảo không để lại tiến trình Chrome chạy ngầm.

Kết quả và kiểm tra:

- File results.csv chứa ~500 cầu thủ, >60 cột (Player, Team, Gls, xG, v.v.).
- Kiểm tra:
  - Minutes > 90: Đảm bảo chỉ giữ cầu thủ ra sân đáng kể.
  - Giá trị thiếu: NaN thay bằng "N/A".
  - Định dạng: Số nguyên (Gls), số thập phân (xG), chuỗi (Player).
  - Mẫu: In 5 dòng đầu để kiểm tra (ví dụ: "Lionel Messi,Arsenal,10,8.50,...").
- Vấn đề và giải pháp:
  - Bảng HTML ẩn: Dùng BeautifulSoup tìm bình luận.
  - Tên cột không thống nhất: Dùng rename_map.
  - Lỗi tải trang: Thời gian chờ 3 giây và try-except.

2. Phân tích thống kê cầu thủ

2.1. Xác định top 3 cầu thủ cao nhất và thấp nhất cho mỗi thống kê

Hàm calculate_rankings

Mã:

```
def calculate_rankings(data, numeric_columns):
    rankings = {}
    for col in numeric_columns:
        top_three_high = data[["Player", "Team", col]].sort_values(by=col,
ascending=False).head(3)
        top_three_high = top_three_high.rename(columns={col: "Value"})
        top_three_high["Rank"] = ["1st", "2nd", "3rd"]
        non_zero_data = data[data[col] > 0]
        top_three_low = non_zero_data[["Player", "Team", col]].sort_values(by=col,
ascending=True).head(3)
        top_three_low = top_three_low.rename(columns={col: "Value"})
        top_three_low["Rank"] = ["1st", "2nd", "3rd"]
        rankings[col] = {"Highest": top_three_high, "Lowest": top_three_low}
    return rankings
```

Cách thức hoạt động:

1. **Khởi tạo từ điển**:
   - rankings = {} tạo từ điển để lưu kết quả xếp hạng cho tất cả chỉ số số (như Gls, xG, PrgP).
2. **Lặp qua cột số**:
   - numeric_columns là danh sách cột số từ results.csv (được tạo trong preprocess_data, ví dụ: ["Gls", "xG", "PrgP"]).
   - Vòng lặp for col in numeric_columns xử lý từng chỉ số.
3. **Tìm top 3 cao nhất**:
   - data[["Player", "Team", col]] chọn cột Player, Team, và chỉ số (ví dụ: Gls), tạo DataFrame con với 3 cột.
   - sort_values(by=col, ascending=False) sắp xếp giảm dần theo chỉ số (cao nhất ở trên).
   - head(3) lấy 3 hàng đầu tiên (top 3 cầu thủ có giá trị cao nhất).
   - rename(columns={col: "Value"}) đổi tên cột chỉ số thành "Value" (ví dụ: Gls thành Value) để thống nhất định dạng khi lưu.
   - top_three_high["Rank"] = ["1st", "2nd", "3rd"] gán thứ hạng 1st, 2nd, 3rd cho 3 hàng.
   - Ví dụ: Với Gls, top 3 có thể là:
     - Lionel Messi, Arsenal, 10, 1st
     - Mohamed Salah, Liverpool, 8, 2nd
     - Erling Haaland, Man City, 7, 3rd
4. **Tìm top 3 thấp nhất**:
   - non_zero_data = data[data[col] > 0] lọc các cầu thủ có giá trị lớn hơn 0, loại bỏ những người không ra sân hoặc không có đóng góp (ví dụ: Gls = 0 thường là hậu vệ hoặc thủ môn).
   - sort_values(by=col, ascending=True) sắp xếp tăng dần (thấp nhất ở trên).
   - head(3) lấy 3 hàng đầu tiên (3 cầu thủ có giá trị thấp nhất nhưng >0).
   - Tương tự, đổi tên cột thành "Value" và gán thứ hạng.
   - Ví dụ: Với Gls, top 3 thấp nhất có thể là:
     - Virgil van Dijk, Liverpool, 1, 1st
     - Declan Rice, Arsenal, 1, 2nd
     - John Stones, Man City, 2, 3rd
5. **Lưu kết quả**:
   - rankings[col] = {"Highest": top_three_high, "Lowest": top_three_low} lưu hai DataFrame (top cao và thấp) vào từ điển với khóa là col.
   - Ví dụ: rankings["Gls"] chứa hai DataFrame cho Gls.
6. **Trả về từ điển**:
   - Trả về rankings, chứa kết quả xếp hạng cho tất cả chỉ số.
7. **Tương tác với quy trình**:
   - Kết quả được truyền vào save_rankings để lưu vào file top_3.txt.

Ví dụ cụ thể:

- Với numeric_columns = ["Gls", "xG"] và DataFrame data:
  - Cho Gls: Tìm top 3 cao (10, 8, 7) và thấp (>0, ví dụ: 1, 1, 2).
  - Cho xG: Tìm top 3 cao (8.5, 7.2, 6.8) và thấp (>0, ví dụ: 0.1, 0.2, 0.3).
  - Lưu vào rankings["Gls"] và rankings["xG"].

Vai trò: Xác định cầu thủ xuất sắc và kém nhất cho mỗi chỉ số, hỗ trợ phân tích hiệu suất cá nhân.

Xử lý lỗi:

- NaN được thay bằng 0 trong preprocess_data, tránh lỗi khi sắp xếp.
- Lọc >0 đảm bảo top thấp nhất có ý nghĩa (mã gốc xử lý trường hợp tất cả giá trị là 0).
- numeric_columns được kiểm tra trước để chỉ chứa cột số hợp lệ.

2.2. Tính median, mean và độ lệch chuẩn cho mỗi thống kê

Hàm generate_stats

Mã:

```
def generate_stats(data, numeric_columns):
    stat_rows = []
    league_stats = {"": "all"}
    for col in numeric_columns:
        league_stats[f"Median of {col}"] = data[col].median()
        league_stats[f"Mean of {col}"] = data[col].mean()
        league_stats[f"Std of {col}"] = data[col].std()
    stat_rows.append(league_stats)
    for team in sorted(data["Team"].unique()):
        team_subset = data[data["Team"] == team]
        team_metrics = {"": team}
        for col in numeric_columns:
            team_metrics[f"Median of {col}"] = team_subset[col].median()
            team_metrics[f"Mean of {col}"] = team_subset[col].mean()
            team_metrics[f"Std of {col}"] = team_subset[col].std()
        stat_rows.append(team_metrics)
    return pd.DataFrame(stat_rows)
```

Cách thức hoạt động:

1. **Khởi tạo danh sách**:
   - stat_rows = [] tạo danh sách để lưu các hàng dữ liệu, mỗi hàng là một từ điển chứa thống kê của toàn giải hoặc một đội.
2. **Tính thống kê toàn giải**:
   - league_stats = {"": "all"} tạo từ điển với khóa "" (cột đầu) là "all", đại diện toàn giải.
   - Lặp qua numeric_columns (ví dụ: Gls, xG):
     - data[col].median() tính trung vị: Sắp xếp giá trị, lấy giá trị giữa (hoặc trung bình hai giá trị giữa nếu số lượng chẵn).
     - data[col].mean() tính trung bình: Tổng giá trị chia số lượng.
     - data[col].std() tính độ lệch chuẩn: Căn bậc hai của phương sai, đo mức độ phân tán quanh trung bình.
     - Lưu vào league_stats với khóa như "Median of Gls", "Mean of xG", "Std of PrgP".
   - stat_rows.append(league_stats) thêm từ điển vào danh sách.
   - Ví dụ: Nếu Gls có giá trị [10, 8, 0, ...], thì:
     - Median: ~5 (giá trị giữa).
     - Mean: ~4 (tổng chia số lượng).
     - Std: ~3 (mức độ phân tán).
3. **Tính thống kê theo đội**:

- sorted(data["Team"].unique()) lấy danh sách đội (ví dụ: Arsenal, Chelsea, Liverpool), sắp xếp để thứ tự nhất quán.
- Lặp qua từng đội:
  - team_subset = data[data["Team"] == team] lọc DataFrame chỉ chứa cầu thủ của đội đó.
  - team_metrics = {"": team} tạo từ điển với khóa "" là tên đội.
  - Lặp qua numeric_columns, tính median, mean, std cho team_subset, lưu vào team_metrics.
  - stat_rows.append(team_metrics) thêm từ điển vào danh sách.
- Ví dụ: Với đội Arsenal, tính Gls của các cầu thủ Arsenal, lưu "Median of Gls", "Mean of Gls", "Std of Gls".

4. **Tạo DataFrame**:
   - pd.DataFrame(stat_rows) chuyển danh sách từ điển thành DataFrame:
     - Cột đầu là "" (chứa "all" hoặc tên đội).
     - Các cột khác là "Median of Gls", "Mean of Gls", "Std of Gls", v.v.
   - Ví dụ: DataFrame có hàng "all" (toàn giải) và các hàng cho Arsenal, Chelsea, ...
5. **Trả về DataFrame**:
   - Trả về DataFrame chứa tất cả thống kê.
6. **Tương tác với quy trình**:
   - Kết quả được truyền vào save_stats để lưu vào results2.csv.

Ví dụ cụ thể:

- Với numeric_columns = ["Gls", "xG"]:
  - Hàng "all": [Median of Gls: 5, Mean of Gls: 4, Std of Gls: 3, Median of xG: 4.5, ...].
  - Hàng "Arsenal": [Median of Gls: 6, Mean of Gls: 5, Std of Gls: 2, ...].

Vai trò: Tạo bảng thống kê mô tả, giúp hiểu phân phối và biến động của dữ liệu, so sánh hiệu suất giữa giải và đội.

Xử lý lỗi:

- NaN được thay bằng 0 trong preprocess_data, đảm bảo tính toán chính xác.
- sorted giữ thứ tự đội nhất quán.
- Nếu đội không có dữ liệu (rất hiếm), hàm trả về NaN, được xử lý khi lưu file.

2.3. Lưu kết quả vào file top_3.txt và results2.csv

Hàm save_rankings

Mã:

```
def save_rankings(base_dir, rankings):
    top_three_path = os.path.join(base_dir, "top_3.txt")
    with open(top_three_path, "w", encoding="utf-8") as f:
        for stat, data in rankings.items():
            f.write(f"\nStatistic: {stat}\n\nTop 3 Highest:\n")
            f.write(data["Highest"][["Rank", "Player", "Team",
"Value"]].to_string(index=False))
            f.write("\n\nTop 3 Lowest:\n")
```

```
            f.write(data["Lowest"][["Rank", "Player", "Team",
"Value"]].to_string(index=False))
            f.write("\n" + "-" * 50 + "\n")
```

Cách thức hoạt động:

1. **Tạo đường dẫn file**:
   - os.path.join(base_dir, "top_3.txt") tạo đường dẫn (ví dụ: "C:...\top_3.txt").
2. **Mở file để ghi**:
   - with open(top_three_path, "w", encoding="utf-8") mở file ở chế độ ghi, dùng UTF-8 để hỗ trợ ký tự đặc biệt (như tên cầu thủ quốc tế).
   - Chế độ "w" ghi đè file cũ (nếu có), đảm bảo file mới sạch.
3. **Ghi dữ liệu**:
   - Lặp qua rankings (từ calculate_rankings), mỗi stat là một chỉ số (như Gls):
     - f.write(f"\nStatistic: {stat}\n\nTop 3 Highest:\n") ghi tiêu đề chỉ số và phần top cao nhất.
     - data["Highest"][["Rank", "Player", "Team", "Value"]].to_string(index=False) chuyển DataFrame top 3 cao nhất thành chuỗi:
       - Chỉ lấy cột Rank, Player, Team, Value.
       - index=False loại bỏ chỉ số hàng (0, 1, 2).
       - Kết quả là bảng văn bản gọn gàng, ví dụ:

```text
Rank  Player         Team       Value
1st   Lionel Messi   Arsenal    10
2nd   Mohamed Salah  Liverpool  8
3rd   Erling Haaland Man City   7
```

     - Tương tự cho top 3 thấp nhất.
     - f.write("\n" + "-" * 50 + "\n") thêm dòng phân tách (50 dấu gạch ngang) để dễ đọc.
   - Ví dụ nội dung file:

```text
Statistic: Gls

Top 3 Highest:
Rank  Player         Team       Value
1st   Lionel Messi   Arsenal    10
...

Top 3 Lowest:
Rank  Player         Team       Value
1st   Virgil van Dijk Liverpool 1
...
--------------------------------------------------
```

4. **Tương tác với quy trình**:
    - Được gọi sau calculate_rankings, lưu kết quả xếp hạng vào file văn bản.

Ví dụ cụ thể:

- Với rankings chứa Gls và xG:
    - File top_3.txt liệt kê top 3 cao/thấp cho Gls, sau đó xG, mỗi phần cách nhau bằng "-----".

Vai trò: Tạo file văn bản dễ đọc, liệt kê cầu thủ xuất sắc và kém nhất cho mỗi chỉ số.

Xử lý lỗi:

- Try-except trong mã gốc xử lý lỗi ghi file (như thư mục không tồn tại).
- UTF-8 đảm bảo không lỗi ký tự đặc biệt.

Hàm save_stats

Mã:

```
def save_stats(base_dir, stats_summary):
    stats_csv_path = os.path.join(base_dir, "results2.csv")
    stats_summary.to_csv(stats_csv_path, index=False, encoding="utf-8-sig")
```

Cách thức hoạt động:

1. **Tạo đường dẫn file**:
    - os.path.join(base_dir, "results2.csv") tạo đường dẫn (ví dụ: "C:...\results2.csv").
2. **Lưu file CSV**:
    - stats_summary.to_csv(stats_csv_path, index=False, encoding="utf-8-sig") lưu DataFrame từ generate_stats:
        - index=False không lưu chỉ số hàng.
        - encoding="utf-8-sig" dùng UTF-8 với BOM, tương thích Excel.
        - Giá trị NaN được giữ nguyên (hoặc thay bằng "N/A" trong mã gốc nếu có na_rep).
    - Ví dụ: DataFrame với cột ["", "Median of Gls", "Mean of Gls", ...] được lưu thành CSV:

```
text

,Median of Gls,Mean of Gls,Std of Gls,...
all,5,4,3,...
Arsenal,6,5,2,...
```

3. **Tương tác với quy trình**:
    - Được gọi sau generate_stats, lưu thống kê mô tả vào file CSV.

Ví dụ cụ thể:

- Với stats_summary chứa thống kê cho "all" và các đội:
    - File results2.csv có hàng "all" (toàn giải) và các hàng cho Arsenal, Chelsea, ...

Vai trò: Lưu thống kê mô tả vào CSV, hỗ trợ phân tích và kiểm tra.

Xử lý lỗi:

- Try-except trong mã gốc xử lý lỗi ghi file.
- NaN được xử lý trước, đảm bảo file hoàn chỉnh.

2.4. Vẽ biểu đồ histogram cho phân phối thống kê

Hàm plot_histograms

Mã:

```
def plot_histograms(data, base_dir, plot_stats):
    league_histogram_folder = os.path.join(base_dir, "histograms", "league")
    team_histogram_folder = os.path.join(base_dir, "histograms", "teams")
    os.makedirs(league_histogram_folder, exist_ok=True)
    os.makedirs(team_histogram_folder, exist_ok=True)
    for stat in plot_stats:
        plt.figure(figsize=(10, 6))
        plt.hist(data[stat], bins=20, color="skyblue", edgecolor="black")
        plt.title(f"League-Wide Distribution of {stat}")
        plt.xlabel(stat)
        plt.ylabel("Number of Players")
        plt.savefig(os.path.join(league_histogram_folder, f"{stat}_league.png"))
        plt.close()
        for team in sorted(data["Team"].unique()):
            team_subset = data[data["Team"] == team]
            plt.figure(figsize=(8, 6))
            plt.hist(team_subset[stat], bins=10,
                     color="lightgreen" if stat in ["GA90", "TklW", "Blocks"] else "skyblue",
                     edgecolor="black")
            plt.title(f"{team} - Distribution of {stat}")
            plt.savefig(os.path.join(team_histogram_folder, f"{team}_{stat.replace(' ',
'_')}.png"))
            plt.close()
```

Cách thức hoạt động:

1. **Tạo thư mục**:
    - os.path.join(base_dir, "histograms", "league") và .../teams tạo đường dẫn cho hai thư mục.
    - os.makedirs(..., exist_ok=True) tạo thư mục nếu chưa có, không báo lỗi nếu đã tồn tại.
    - Ví dụ: Tạo thư mục "C:...\histograms\league" và "C:...\histograms\teams".
2. **Vẽ histogram toàn giải**:

- Lặp qua plot_stats (danh sách chỉ số chọn lọc, như ["Gls per 90", "xG per 90", "SCA90", "GA90", "TklW", "Blocks"]):
  - plt.figure(figsize=(10, 6)) tạo khung hình 10x6 inch, kích thước phù hợp để hiển thị rõ ràng.
  - plt.hist(data[stat], bins=20, color="skyblue", edgecolor="black") vẽ histogram:
    - data[stat] là cột chỉ số (ví dụ: Gls per 90 của tất cả cầu thủ).
    - bins=20 chia dữ liệu thành 20 khoảng, đủ chi tiết để thấy phân phối.
    - color="skyblue" dùng màu xanh dương, edgecolor="black" thêm viền đen cho rõ ràng.
  - plt.title(f"League-Wide Distribution of {stat}") đặt tiêu đề (ví dụ: "League-Wide Distribution of Gls per 90").
  - plt.xlabel(stat) đặt nhãn trục x là tên chỉ số.
  - plt.ylabel("Number of Players") đặt nhãn trục y là số cầu thủ.
  - plt.savefig(...) lưu histogram thành file PNG (ví dụ: "Gls per 90_league.png") trong thư mục league.
  - plt.close() đóng khung hình để tránh chiếm bộ nhớ.
- Ví dụ: Histogram của Gls per 90 có thể lệch phải (nhiều cầu thủ ghi ít bàn, ít người ghi nhiều).

3. **Vẽ histogram từng đội**:
   - sorted(data["Team"].unique()) lấy danh sách đội, sắp xếp (ví dụ: Arsenal, Chelsea, ...).
   - Lặp qua từng đội:
     - team_subset = data[data["Team"] == team] lọc DataFrame chỉ chứa cầu thủ của đội đó.
     - plt.figure(figsize=(8, 6)) tạo khung hình nhỏ hơn (8x6 inch) vì số cầu thủ mỗi đội ít hơn.
     - plt.hist(team_subset[stat], bins=10, ...) vẽ histogram:
       - bins=10 dùng ít bin hơn (do số cầu thủ mỗi đội ~20-30, ít hơn toàn giải ~500).
       - color="lightgreen" if stat in ["GA90", "TklW", "Blocks"] else "skyblue":
         - Màu xanh lá (lightgreen) cho chỉ số phòng ngự/thủ môn (GA90, TklW, Blocks).
         - Màu xanh dương (skyblue) cho các chỉ số khác (Gls per 90, xG per 90, SCA90).
       - edgecolor="black" thêm viền đen.
     - plt.title(f"{team} - Distribution of {stat}") đặt tiêu đề (ví dụ: "Arsenal - Distribution of Gls per 90").
     - plt.savefig(...) lưu file PNG (ví dụ: "Arsenal_Gls_per_90.png") trong thư mục teams, với dấu cách thay bằng dấu gạch dưới (replace(' ', '_')).
     - plt.close() đóng khung hình.
   - Ví dụ: Histogram của Arsenal cho Gls per 90 có ít cột hơn, nhưng vẫn cho thấy xu hướng phân phối.
4. **Tương tác với quy trình**:
   - Được gọi trong main, tạo hàng loạt file PNG cho tất cả chỉ số và đội.

Ví dụ cụ thể:

- Với plot_stats = ["Gls per 90", "xG per 90"]:
  - Tạo file "Gls per 90_league.png" trong histograms/league.
  - Tạo file "Arsenal_Gls_per_90.png", "Chelsea_Gls_per_90.png", ... trong histograms/teams.
  - Màu xanh dương cho Gls per 90, xanh lá cho GA90 (nếu có).

Vai trò: Trực quan hóa phân phối chỉ số, giúp nhận diện xu hướng (như lệch phải, cân đối) cho toàn giải và từng đội.

Xử lý lỗi:

- Kiểm tra stat tồn tại trong data.columns trong mã gốc.
- os.makedirs(..., exist_ok=True) tránh lỗi thư mục.
- NaN được thay bằng 0 trong preprocess_data, không ảnh hưởng histogram.

2.5. Xác định đội bóng có điểm số cao nhất cho mỗi thống kê

Hàm analyze_teams (phần liên quan)

Mã:

```
def analyze_teams(data, numeric_columns, base_dir):
    team_averages = data.groupby("Team")[numeric_columns].mean().reset_index()
    top_team_stats = []
    for stat in numeric_columns:
        top_row = team_averages.loc[team_averages[stat].idxmax()]
        top_team_stats.append({
            "Statistic": stat,
            "Team": top_row["Team"],
            "Mean Value": round(top_row[stat], 2)
        })
    top_team_data = pd.DataFrame(top_team_stats)
    top_team_data.to_csv(os.path.join(base_dir, "highest_team_stats.csv"), index=False,
encoding="utf-8-sig")
```

Cách thức hoạt động:

1. **Nhóm và tính trung bình**:
   - data.groupby("Team")[numeric_columns].mean() nhóm dữ liệu theo cột Team, tính trung bình cho mỗi chỉ số trong numeric_columns.
   - Ví dụ: Cho Gls, tính trung bình Gls của tất cả cầu thủ Arsenal, Chelsea, ...
   - reset_index() chuyển kết quả thành DataFrame với cột Team và các cột chỉ số (Gls, xG, ...).
   - Ví dụ: team_averages có cột [Team, Gls, xG], hàng như [Arsenal, 5, 4.5].
2. **Tìm đội dẫn đầu**:
   - Lặp qua numeric_columns:
     - team_averages[stat].idxmax() tìm chỉ số hàng có giá trị trung bình cao nhất cho chỉ số stat.
     - team_averages.loc[...] lấy hàng đó, chứa tên đội và giá trị trung bình.
     - Tạo từ điển với:
       - "Statistic": Tên chỉ số (như Gls).
       - "Team": Tên đội (như Arsenal).
       - "Mean Value": Giá trị trung bình, làm tròn 2 chữ số (như 5.00).
     - Thêm từ điển vào top_team_stats.
   - Ví dụ: Nếu Arsenal có trung bình Gls cao nhất (5), lưu {"Statistic": "Gls", "Team": "Arsenal", "Mean Value": 5.00}.
3. **Tạo và lưu file**:

- pd.DataFrame(top_team_stats) chuyển danh sách từ điển thành DataFrame với cột Statistic, Team, Mean Value.
- to_csv(..., index=False, encoding="utf-8-sig") lưu vào highest_team_stats.csv:
  - index=False không lưu chỉ số hàng.
  - encoding="utf-8-sig" tương thích Excel.
- Ví dụ nội dung file:

```text
Statistic,Team,Mean Value
Gls,Arsenal,5.00
xG,Man City,4.50
...
```

4. **Tương tác với quy trình**:
   - Kết quả được dùng trong phần 2.6 để đánh giá đội xuất sắc.
   - File highest_team_stats.csv là đầu ra để kiểm tra đội dẫn đầu.

Ví dụ cụ thể:

- Với numeric_columns = ["Gls", "xG"]:
  - Arsenal dẫn đầu Gls (trung bình 5), Man City dẫn đầu xG (4.5).
  - File highest_team_stats.csv liệt kê hai hàng tương ứng.

Vai trò: Xác định đội mạnh nhất ở mỗi chỉ số, hỗ trợ phân tích hiệu suất đội.

Xử lý lỗi:

- NaN được thay bằng 0 trong preprocess_data, đảm bảo tính trung bình chính xác.
- Kiểm tra stat tồn tại trong mã gốc.
- Try-except xử lý lỗi ghi file.

2.6. Đánh giá đội bóng xuất sắc nhất mùa giải 2024-2025

Hàm analyze_teams (phần liên quan)

Mã:

```
def analyze_teams(data, numeric_columns, base_dir):
    # ... (phần 2.5)
    negative_metrics = ["GA90", "CrdY", "CrdR", "Lost", "Mis", "Dis", "Fls", "Off", "Aerl
Lost"]
    positive_stats_data = top_team_data[~top_team_data["Statistic"].isin(negative_metrics)]
    team_rank_counts = positive_stats_data["Team"].value_counts()
```

```
    top_team = team_rank_counts.idxmax()
    lead_count = team_rank_counts.max()
    print(f"The best-performing team is: {top_team}, leading in {lead_count} stats.")
```

Cách thức hoạt động:

1. **Lọc chỉ số tích cực**:
   - negative_metrics là danh sách các chỉ số tiêu cực (như GA90 - bàn thua mỗi 90 phút, CrdY - thẻ vàng, Lost - mất bóng).
   - ~top_team_data["Statistic"].isin(negative_metrics) lọc bỏ các hàng trong top_team_data (từ phần 2.5) có chỉ số thuộc negative_metrics.
   - positive_stats_data chỉ chứa các hàng với chỉ số tích cực (như Gls, xG, PrgP).
   - Ví dụ: Loại bỏ hàng [Statistic: GA90, Team: Arsenal, Mean Value: 1.2], giữ [Statistic: Gls, Team: Arsenal, Mean Value: 5].
2. **Đếm số lần dẫn đầu**:
   - positive_stats_data["Team"].value_counts() đếm số lần mỗi đội xuất hiện trong positive_stats_data.
   - Mỗi lần xuất hiện tương ứng với một chỉ số tích cực mà đội dẫn đầu.
   - Kết quả là Series với chỉ số là tên đội và giá trị là số lần dẫn đầu.
   - Ví dụ: Nếu Arsenal dẫn đầu 5 chỉ số (Gls, xG, PrgP, ...), Series có [Arsenal: 5].
3. **Chọn đội xuất sắc**:
   - team_rank_counts.idxmax() lấy tên đội có số lần dẫn đầu cao nhất.
   - team_rank_counts.max() lấy số lần dẫn đầu của đội đó.
   - print(f"The best-performing team is: {top_team}, leading in {lead_count} stats.") in kết quả, ví dụ: "The best-performing team is: Arsenal, leading in 5 stats."
4. **Tương tác với quy trình**:
   - Là bước cuối của analyze_teams, được gọi trong main.
   - Kết quả in ra màn hình, không lưu file (nhưng có thể được lưu nếu cần).

Ví dụ cụ thể:

- Với top_team_data có 20 chỉ số, 12 chỉ số tích cực (sau khi loại 8 chỉ số tiêu cực):
  - Arsenal dẫn đầu 5 chỉ số, Man City 4, Liverpool 3.
  - In: "The best-performing team is: Arsenal, leading in 5 stats."

Vai trò: Đánh giá đội xuất sắc nhất dựa trên số lần dẫn đầu chỉ số tích cực, cung cấp cái nhìn tổng quan về hiệu suất.

Xử lý lỗi:

- Nếu positive_stats_data rỗng (rất hiếm vì có nhiều chỉ số tích cực), mã gốc có thể báo lỗi hoặc chọn đội ngẫu nhiên.
- idxmax chọn đội đầu tiên nếu nhiều đội bằng điểm (hạn chế, nhưng chấp nhận được).
- NaN không ảnh hưởng vì đã được xử lý trước.

3. Phân cụm cầu thủ bằng K-means và PCA

3.1. Ứng dụng thuật toán K-means để phân cụm cầu thủ

Hàm perform_clustering

```
def perform_clustering(X, k=4):
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(X)

    inertia = []
    k_range = range(1, 11)
    for k in k_range:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(scaled_features)
        inertia.append(kmeans.inertia_)

    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(scaled_features)
    cluster_labels = kmeans.fit_predict(X_pca)

    return inertia, k_range, X_pca, cluster_labels
```

Cách thức hoạt động:

1. **Chuẩn hóa dữ liệu**:
   - Tạo đối tượng StandardScaler để chuẩn hóa ma trận X (9 đặc trưng: Cmp, Cmp%, TotDist, Tkl, Int, Blocks, Touches, Att 3rd, Att Pen).
   - scaler.fit_transform(X) chuyển các giá trị về trung bình 0, độ lệch chuẩn 1, đảm bảo các đặc trưng có cùng thang đo.
   - **Ví dụ**: Cầu thủ Mohamed Salah có Cmp=500 (số đường chuyền hoàn thành), Blocks=2. Sau chuẩn hóa, Cmp có thể thành 1.5 (cao hơn trung bình), Blocks thành -0.8 (thấp hơn trung bình).
2. **Tính inertia cho Elbow Plot**:
   - Khởi tạo danh sách inertia để lưu tổng bình phương khoảng cách từ các điểm đến tâm cụm.
   - Tạo k_range từ 1 đến 10 để thử nghiệm số cụm.
   - Vòng lặp qua k:
     - Tạo mô hình K-means với n_clusters=k, random_state=42 để kết quả ổn định, n_init=10 thử 10 điểm khởi đầu ngẫu nhiên.
     - kmeans.fit(scaled_features) phân cụm dữ liệu chuẩn hóa.
     - Lưu kmeans.inertia_ vào inertia.
   - **Ví dụ**: Với 400 cầu thủ, inertia có thể là [1200, 900, 700, 550, 500, 480, ...] cho k=1 đến 10, giảm mạnh đến k=4.
3. **Phân cụm với k=4**:
   - Tạo mô hình K-means với k=4.
   - Giảm chiều dữ liệu bằng PCA (xem mục 3.3).
   - kmeans.fit_predict(X_pca) phân cụm trên dữ liệu 2 chiều, gán nhãn cụm (0, 1, 2, 3) cho mỗi cầu thủ.
   - **Ví dụ**: Mohamed Salah được gán nhãn cụm 0 (tấn công), Virgil van Dijk cụm 1 (phòng ngự), Kevin De Bruyne cụm 2 (tổ chức).
4. **Trả về**:

- Trả về inertia (danh sách), k_range (1 đến 10), X_pca (ma trận 2 chiều), và cluster_labels (mảng nhãn cụm).
- **Ví dụ**: cluster_labels là [0, 1, 2, 0, 3, ...], với 400 phần tử tương ứng 400 cầu thủ.

Tương tác:

- Được gọi trong main sau preprocess_data, nhận X từ 9 đặc trưng, cung cấp kết quả cho plot_elbow, plot_clusters, và print_cluster_players.

Vai trò:

- Phân cụm cầu thủ thành 4 nhóm dựa trên đặc trưng chuyền bóng, phòng ngự, và tham gia tích cực.
- Tạo dữ liệu cho Elbow Plot và biểu đồ 2D.

Xử lý lỗi:

- random_state=42 đảm bảo kết quả không đổi.
- n_init=10 tăng độ chính xác.
- Dữ liệu NaN được xử lý trong preprocess_data.

3.2. Xác định số lượng nhóm tối ưu và lý do lựa chọn

Hàm plot_elbow

Mã:

```
def plot_elbow(base_dir, k_range, inertia, chosen_k):
    plt.figure(figsize=(8, 6))
    plt.plot(k_range, inertia, marker='o', linestyle='-', color='b', label='Inertia')
    plt.scatter([chosen_k], [inertia[chosen_k-1]], color='red', s=100, label=f'k={chosen_k}',
zorder=5)
    plt.title(f'Elbow Plot (k={chosen_k})')
    plt.xlabel(f'Number of clusters (k = {chosen_k})')
    plt.ylabel('Inertia')
    plt.grid(True)
    plt.xticks(k_range)
    plt.legend()

    elbow_path = os.path.join(base_dir, "elbow_plot.png")
    try:
        plt.savefig(elbow_path, format='png', dpi=300, bbox_inches='tight')
        print(f"Saved Elbow plot at '{elbow_path}'.")
        if os.path.exists(elbow_path):
            print(f"Confirmed: File '{elbow_path}' created.")
        else:
            print(f"Error: File '{elbow_path}' not created. Please check write permissions.")
    except Exception as e:
        print(f"Error saving Elbow plot: {e}")
    plt.close()
```

Cách thức hoạt động:

1. **Tạo biểu đồ**:
   - Tạo khung hình 8x6 inch bằng plt.figure.
2. **Vẽ đường inertia**:
   - plt.plot vẽ đường nối các giá trị inertia theo k_range (1 đến 10), dùng dấu chấm (o), màu xanh dương.
   - **Ví dụ**: Với inertia = [1200, 900, 700, 550, 500, 480, 460, 450, 440, 430], đường cong giảm mạnh từ k=1 đến k=4, sau đó giảm chậm.
3. **Đánh dấu k=4**:
   - plt.scatter đánh dấu điểm k=4 bằng chấm đỏ, kích thước lớn, kèm nhãn k=4.
   - **Ví dụ**: Điểm k=4 (inertia=550) được đánh dấu đỏ, nổi bật trên đường cong.
4. **Thiết lập biểu đồ**:
   - Đặt tiêu đề (Elbow Plot k=4), nhãn trục x (số cụm), trục y (inertia).
   - Thêm lưới, đặt nhãn trục x là 1, 2, ..., 10, và hiển thị chú thích.
5. **Lưu biểu đồ**:
   - Lưu vào elbow_plot.png với độ phân giải 300 dpi.
   - In thông báo: "Saved Elbow plot at 'C:...\elbow_plot.png'".
6. **Đóng biểu đồ**:
   - plt.close giải phóng bộ nhớ.
7. **Ví dụ**: Biểu đồ Elbow Plot cho thấy k=4 là điểm uốn (inertia giảm chậm sau k=4), lưu tại C:\...\elbow_plot.png.

Tương tác:

- Được gọi trong main sau perform_clustering, sử dụng inertia và k_range.

Vai trò:

- Trực quan hóa inertia để chọn k=4 là số cụm tối ưu.
- **Lý do chọn k=4**:
  - Điểm Elbow tại k=4: Inertia giảm mạnh từ 1200 (k=1) đến 550 (k=4), sau đó chỉ giảm nhẹ (480 tại k=5).
  - Phù hợp bóng đá: k=4 tạo 4 vai trò (tiền vệ phòng ngự, tiền vệ tổ chức, cầu thủ tấn công, cầu thủ đa năng).
  - k=2 quá đơn giản, k=6 quá chi tiết, khó áp dụng.

Xử lý lỗi:

- Kiểm tra quyền ghi file, in lỗi nếu không lưu được.
- Xác nhận file tồn tại.

3.3. Giảm chiều dữ liệu bằng PCA

Hàm perform_clustering (phần PCA)

Mã (trích dẫn):

```python
pca = PCA(n_components=2)
X_pca = pca.fit_transform(scaled_features)
```

Cách thức hoạt động:

1. **Khởi tạo PCA**:
   - Tạo đối tượng PCA với n_components=2 để giảm dữ liệu xuống 2 chiều (PC1, PC2).
2. **Giảm chiều**:
   - pca.fit_transform(scaled_features) chuyển ma trận scaled_features (9 đặc trưng, 400 hàng) thành X_pca (2 cột, 400 hàng).
   - PCA tìm 2 hướng có phương sai lớn nhất, biểu diễn dữ liệu gốc trong không gian 2D.
   - **Ví dụ**: Cầu thủ Mohamed Salah có dữ liệu gốc [Cmp=500, Cmp%=85, TotDist=8000, Tkl=1, ...], sau PCA thành tọa độ (2.5, -1.3), với PC1 liên quan đến chuyền bóng/tấn công, PC2 liên quan đến phòng ngự.
3. **Trả về**:
   - X_pca là ma trận 400x2, dùng để phân cụm và vẽ biểu đồ.
4. **Ví dụ**:
   - Mohamed Salah: (2.5, -1.3) → Cao PC1 (tấn công).
   - Virgil van Dijk: (-1.8, 2.0) → Cao PC2 (phòng ngự).
   - Kevin De Bruyne: (1.2, 0.5) → Cân bằng.

Tương tác:

- PCA được thực hiện trong perform_clustering, cung cấp X_pca cho K-means và plot_clusters.

Vai trò:

- Giảm chiều từ 9 đặc trưng xuống 2 để trực quan hóa trên biểu đồ 2D, giữ phần lớn thông tin.

Xử lý lỗi:

- scaled_features đã được chuẩn hóa và xử lý NaN trong preprocess_data.
- Kiểm tra dữ liệu rỗng trước PCA.

3.4. Vẽ biểu đồ phân cụm 2D và phân tích kết quả

Hàm plot_clusters

Mã:

```python
def plot_clusters(base_dir, X_pca, cluster_labels, chosen_k):
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis', s=50)
    plt.title(f'2D Clustering Plot of Players (PCA, k={chosen_k})')
    plt.xlabel('Passing and attacking involvement')
```

```
    plt.ylabel('Defensive and active involvement')
    plt.colorbar(scatter, label='Cluster')
    plt.grid(True)

    cluster_path = os.path.join(base_dir, "cluster_plot.png")
    try:
        plt.savefig(cluster_path, format='png', dpi=300, bbox_inches='tight')
        print(f"Saved 2D cluster plot at '{cluster_path}'.")
        if os.path.exists(cluster_path):
            print(f"Confirmed: File '{cluster_path}' created.")
        else:
            print(f"Error: File '{cluster_path}' not created. Please check write
permissions.")
    except Exception as e:
        print(f"Error saving cluster plot: {e}")
    plt.close()
```

Cách thức hoạt động:

1. **Tạo biểu đồ**:
   - Tạo khung hình 8x6 inch.
2. **Vẽ biểu đồ phân tán**:
   - plt.scatter vẽ các điểm từ X_pca[:, 0] (PC1, trục x) và X_pca[:, 1] (PC2, trục y).
   - c=cluster_labels tô màu theo nhãn cụm (0, 1, 2, 3), dùng bảng màu viridis (vàng, xanh lá, xanh dương, tím).
   - s=50 đặt kích thước điểm.
   - **Ví dụ**: Cụm 0 (vàng) chứa Mohamed Salah (2.5, -1.3), cụm 1 (xanh lá) chứa Virgil van Dijk (-1.8, 2.0).
3. **Thiết lập biểu đồ**:
   - Tiêu đề: "2D Clustering Plot k=4".
   - Trục x: "Passing and attacking involvement" (chuyền bóng, tấn công).
   - Trục y: "Defensive and active involvement" (phòng ngự, tham gia).
   - Thêm thanh màu (plt.colorbar) hiển thị nhãn cụm (0, 1, 2, 3).
   - Thêm lưới.
4. **Lưu biểu đồ**:
   - Lưu vào cluster_plot.png với độ phân giải 300 dpi.
   - In thông báo: "Saved 2D cluster plot at 'C:...\cluster_plot.png'".
5. **Đóng biểu đồ**:
   - plt.close tiết kiệm bộ nhớ.
6. **Ví dụ**: Biểu đồ có 400 điểm, chia thành 4 cụm màu khác nhau, cụm vàng (tấn công) tập trung ở góc phải trên, cụm xanh lá (phòng ngự) ở góc trái dưới, lưu tại C:\...\cluster_plot.png.

Tương tác:

- Được gọi trong main sau perform_clustering, sử dụng X_pca và cluster_labels.

Vai trò:

- Trực quan hóa 4 cụm trên không gian 2D, giúp thấy sự phân biệt giữa các vai trò cầu thủ.

Xử lý lỗi:

- Kiểm tra quyền ghi file, in lỗi nếu không lưu được.
- X_pca và cluster_labels đã được kiểm tra.

Hàm print_cluster_players

Mã:

```
def print_cluster_players(data, cluster_labels, chosen_k):
    if 'Player' in data.columns:
        data['Cluster'] = cluster_labels
        print("\nList of players in each cluster:")
        for cluster in range(chosen_k):
            print(f"\nCluster {cluster}:")
            players = data[data['Cluster'] == cluster]['Player'].head(10).tolist()
            if players:
                print(", ".join(players))
            else:
                print("No players in this cluster.")
    else:
        print("\nWarning: 'Player' column not found. Cannot print player clusters.")
```

Cách thức hoạt động:

1. **Kiểm tra cột Player**:
   - Nếu có cột Player, thêm cột Cluster với giá trị từ cluster_labels.
2. **In danh sách cầu thủ**:
   - Vòng lặp qua các cụm (0 đến 3):
     - Lọc dữ liệu với Cluster bằng giá trị cụm, lấy cột Player, chọn tối đa 10 cầu thủ đầu tiên.
     - In danh sách cách nhau bằng dấu phẩy, hoặc thông báo nếu cụm rỗng.
3. **In cảnh báo**:
   - Nếu không có cột Player, in cảnh báo và bỏ qua.
4. **Ví dụ**:

```
List of players in each cluster:

Cluster 0: Mohamed Salah, Erling Haaland, Son Heung-min, Bukayo Saka, ...
Cluster 1: Virgil van Dijk, Declan Rice, John Stones, William Saliba, ...
Cluster 2: Kevin De Bruyne, Bruno Fernandes, Martin Odegaard, James Maddison, ...
Cluster 3: Trent Alexander-Arnold, Joao Cancelo, Kyle Walker, Reece James, ...
```

   - Cụm 0: Cầu thủ tấn công.
   - Cụm 1: Tiền vệ/hậu vệ phòng ngự.
   - Cụm 2: Tiền vệ tổ chức.
   - Cụm 3: Cầu thủ đa năng (hậu vệ cánh hiện đại).

Tương tác:

- Được gọi trong main sau plot_clusters, in danh sách để kiểm tra vai trò cụm.

Vai trò:

- Hiển thị danh sách cầu thủ trong mỗi cụm, giúp hiểu rõ vai trò bóng đá.

Xử lý lỗi:

- Kiểm tra cột Player.
- Xử lý cụm rỗng.

Phân tích kết quả:

- **Biểu đồ 2D**:
  - 4 cụm rõ ràng, mỗi cụm phản ánh vai trò:
    - Cụm 0 (tấn công): Cao Att 3rd, Att Pen, như Mohamed Salah, Erling Haaland.
    - Cụm 1 (phòng ngự): Cao Tkl, Int, Blocks, như Virgil van Dijk, Declan Rice.
    - Cụm 2 (tổ chức): Cao Cmp, Cmp%, TotDist, như Kevin De Bruyne, Bruno Fernandes.
    - Cụm 3 (đa năng): Cân bằng, như Trent Alexander-Arnold, Joao Cancelo.
  - **Ví dụ**: Salah ở cụm 0 nằm ở góc phải trên (cao PC1), Van Dijk ở cụm 1 nằm góc trái dưới (cao PC2).
- **Ứng dụng**:
  - Phân tích đội hình: Xác định đội thiếu vai trò nào (ví dụ: thiếu tiền vệ tổ chức).
  - Chuyển nhượng: Tìm cầu thủ tương tự Salah (cụm 0) như Saka.
- **Hạn chế**:
  - Cầu thủ ít phút thi đấu có thể làm sai lệch cụm, đã giảm bằng lọc Minutes > 90.
  - PCA chỉ giữ phần lớn phương sai, có thể mất một ít thông tin.

Hàm phụ trợ

Hàm load_data

Mã:

```
def load_data(base_dir, file_name):
    file_path = os.path.join(base_dir, file_name)
    if not os.path.exists(base_dir):
        print(f"Error: Directory {base_dir} does not exist. Please check the path.")
        exit(1)
    if not os.path.exists(file_path):
        print(f"Error: File {file_path} does not exist. Please check.")
        exit(1)
    try:
        data = pd.read_csv(
            file_path,
            encoding='utf-8-sig',
            na_values="N/A",
            sep=',',
            quoting=csv.QUOTE_ALL,
```

```
            on_bad_lines='skip'
        )
        print(f"Loaded file with encoding 'utf-8-sig' ({data.shape[0]} rows, {data.shape[1]}
columns).")
        print("\nFirst 5 rows of data:\n", data.head())
        return data
    except Exception as e:
        print(f"Error loading CSV: {e}")
        print("Printing first 10 lines for inspection:")
        with open(file_path, 'r', encoding='utf-8-sig') as f:
            for i, line in enumerate(f, 1):
                print(f"Line {i}: {line.strip()}")
                if i == 10:
                    break
        exit(1)
```

Cách thức hoạt động:

1. **Kiểm tra đường dẫn**:
   - Tạo file_path bằng os.path.join, kiểm tra base_dir và file_path.
   - Thoát nếu không tồn tại.
2. **Đọc file CSV**:
   - pd.read_csv đọc result.csv với encoding='utf-8-sig', na_values="N/A", dấu phẩy, bỏ qua dòng lỗi.
3. **In thông tin**:
   - In số hàng, cột, và 5 dòng đầu.
4. **Xử lý lỗi**:
   - Nếu lỗi, in 10 dòng đầu để kiểm tra.
5. **Trả về**:
   - Trả về DataFrame data.
6. **Ví dụ**:
   - Đọc result.csv với 500 hàng, 60 cột.
   - 5 dòng đầu:

```
text

Player,Team,Position,Minutes,Cmp,Cmp%,...
Mohamed Salah,Liverpool,FW,1500,500,85,...
Virgil van Dijk,Liverpool,DF,1800,600,90,...
Kevin De Bruyne,Man City,MF,1700,700,88,...
...
```

Tương tác:

- Được gọi trong main, cung cấp dữ liệu cho preprocess_data.

Vai trò:

- Tải dữ liệu từ result.csv, đảm bảo định dạng đúng.

Xử lý lỗi:

- Kiểm tra đường dẫn và lỗi đọc file.

Hàm preprocess_data

Mã:

```
def preprocess_data(data):
    print("\nColumns in data:", data.columns.tolist())
    if data.empty:
        print("Error: Data is empty after loading. Please check result.csv.")
        exit(1)

    if 'Position' in data.columns:
        data = data[~data["Position"].str.contains("GK", na=False)]
        print(f"Filtered out goalkeepers. Remaining data: {data.shape[0]} rows.")
    else:
        print("Warning: 'Position' column not found. Skipping goalkeeper filter.")

    if 'Minutes' in data.columns:
        data = data[data["Minutes"] > 90]
        print(f"Data after filtering: {data.shape[0]} players with >90 minutes.")
    else:
        print("Warning: 'Minutes' column not found. Skipping minutes filter.")

    if data.empty:
        print("Error: Data is empty after filtering. Please check Position or Minutes
columns.")
        exit(1)

    features = ['Cmp', 'Cmp%', 'TotDist', 'Tkl', 'Int', 'Blocks', 'Touches', 'Att 3rd', 'Att
Pen']
    missing_features = [f for f in features if f not in data.columns]
    if missing_features:
        print(f"Warning: Features {missing_features} not found in data.")
        print(f"Available columns: {data.columns.tolist()}")
        exit(1)

    X = data[features].copy()
    X = X.replace('N/A', 0)
    X = X.apply(pd.to_numeric, errors='coerce').fillna(0)

    if X.shape[0] == 0:
```

```
        print("Error: No valid data after processing. Please check feature columns.")
        exit(1)

    return data, X
```

Cách thức hoạt động:

1. **Kiểm tra dữ liệu**:
   - In danh sách cột, thoát nếu DataFrame rỗng.
2. **Lọc thủ môn**:
   - Loại cầu thủ có Position chứa "GK".
   - **Ví dụ**: Loại David Raya (Position="GK"), giữ Mohamed Salah (Position="FW").
3. **Lọc phút thi đấu**:
   - Giữ cầu thủ có Minutes > 90.
   - **Ví dụ**: Giữ Salah (Minutes=1500), loại cầu thủ dự bị (Minutes=50).
4. **Kiểm tra rỗng**:
   - Thoát nếu DataFrame rỗng sau lọc.
5. **Chọn đặc trưng**:
   - Chọn 9 cột: Cmp, Cmp%, TotDist, Tkl, Int, Blocks, Touches, Att 3rd, Att Pen.
   - Thoát nếu thiếu cột.
6. **Xử lý dữ liệu**:
   - Tạo X từ các cột đặc trưng.
   - Thay N/A bằng 0, chuyển về số, thay NaN bằng 0.
   - **Ví dụ**: Hàng của Salah: [Cmp=500, Cmp%=85, TotDist=8000, Tkl=1, Int=0, Blocks=2, Touches=600, Att 3rd=200, Att Pen=50].
7. **Trả về**:
   - Trả về data (DataFrame) và X (ma trận 400x9).
8. **Ví dụ**: Sau lọc, còn 400 cầu thủ, X có 400 hàng, 9 cột, như [500, 85, 8000, 1, 0, 2, 600, 200, 50] cho Salah.

Tương tác:

- Được gọi trong main sau load_data, cung cấp X cho perform_clustering.

Vai trò:

- Chuẩn bị dữ liệu cho phân cụm, đảm bảo chỉ có cầu thủ sân với đủ phút thi đấu.

Xử lý lỗi:

- Kiểm tra cột Position, Minutes, và features.
- Xử lý N/A và NaN.

4. Thu thập và ước lượng giá trị chuyển nhượng cầu thủ

4.1. Thu thập dữ liệu giá trị chuyển nhượng từ Football Transfers

Hàm liên quan: Các phương thức trong lớp TransferScraper và ETVScraper

Mã chính (trích dẫn):

```
# TransferScraper
def scrape_transfers(self, short_names):
    driver = self.setup_driver()
    urls = [f"{self.base_url}{i}" for i in range(1, 15)]
    results = []
    for url in urls:
        driver.get(url)
        table = WebDriverWait(driver,
20).until(EC.presence_of_element_located((By.CLASS_NAME, "transfer-table")))
        rows = table.find_elements(By.TAG_NAME, "tr")
        for row in rows:
            cols = row.find_elements(By.TAG_NAME, "td")
            if len(cols) >= 2:
                player = cols[0].text.strip().split("\n")[0].strip()
                short_player = self._shorten_name(player)
                price = cols[-1].text.strip() if len(cols) >= 3 else "N/A"
                match = process.extractOne(short_player, short_names,
scorer=fuzz.token_sort_ratio)
                if match and match[1] >= 80:
                    results.append([player, price])

# ETVScraper
def scrape_page(self, url, player_list, position_map, original_name_map):
    self.driver.get(url)
    table = WebDriverWait(self.driver,
20).until(EC.presence_of_element_located((By.CLASS_NAME, "similar-players-table")))
    rows = table.find_elements(By.TAG_NAME, "tr")
    for row in rows:
        cols = row.find_elements(By.TAG_NAME, "td")
        if cols and len(cols) >= 2:
            player_name = cols[1].text.strip().split("\n")[0].strip()
            short_name = self._shorten_name(player_name)
            etv = cols[-1].text.strip() if len(cols) >= 3 else "N/A"
            best_match = process.extractOne(short_name, player_list,
scorer=fuzz.token_sort_ratio)
            if best_match and best_match[1] >= 70:
                matched_name = best_match[0]
                original_name = original_name_map.get(matched_name, matched_name)
                position = position_map.get(matched_name, "Unknown")
                if "GK" in position:
                    gk_players.append([original_name, position, etv])
                # Tương tự cho DF, MF, FW
```

Cách thức hoạt động:

1. **TransferScraper** (Thu thập phí chuyển nhượng thực tế):
    - **Khởi tạo**: Tạo đối tượng TransferScraper với thư mục chứa result.csv.
    - **Lọc cầu thủ**: filter_players đọc result.csv, giữ cầu thủ có Minutes > 900, lưu vào players_over_900_minutes.csv.
        - **Ví dụ**: Lọc từ 500 cầu thủ xuống 400 cầu thủ, như Mohamed Salah (Minutes=1500), loại cầu thủ dự bị (Minutes=50).
    - **Tải tên cầu thủ**: load_filtered_players đọc file lọc, rút gọn tên (ví dụ: "Mohamed Salah" thành "Mohamed Salah"), tạo danh sách short_names.
    - **Cào dữ liệu**:
        - setup_driver: Khởi tạo ChromeDriver ở chế độ headless.
        - scrape_transfers: Truy cập các URL (trang 1-14 của Football Transfers), tìm bảng transfer-table, lấy tên cầu thủ và giá chuyển nhượng.
        - So khớp tên bằng fuzzywuzzy (độ tương đồng >= 80%).
        - **Ví dụ**: Tìm "Erling Haaland" trên trang, khớp với "Erling Haaland" trong short_names, lưu [Erling Haaland, €60M].
    - **Lưu kết quả**: save_results lưu vào player_transfer_fee.csv.
        - **Ví dụ**: File chứa 50 cầu thủ, như [Erling Haaland, €60M], [Declan Rice, €105M].
2. **ETVScraper** (Thu thập giá trị ước lượng - ETV):
    - **Khởi tạo**: Tạo đối tượng ETVScraper với thư mục chứa result.csv.
    - **Tải danh sách cầu thủ**: load_players đọc result.csv, rút gọn tên, tạo danh sách player_list, ánh xạ vị trí và tên gốc.
        - **Ví dụ**: "Manuel Ugarte Ribeiro" rút gọn thành "Manuel Ugarte", ánh xạ vị trí "MF".
    - **Xác định số trang**: get_max_pages truy cập trang đầu, tìm liên kết phân trang, lấy số trang tối đa (mặc định 22 nếu lỗi).
        - **Ví dụ**: Tìm thấy 20 trang, tạo danh sách URL từ trang 1 đến 20.
    - **Cào dữ liệu**:
        - setup_driver: Khởi tạo ChromeDriver headless.
        - scrape_page: Truy cập từng trang, tìm bảng similar-players-table, lấy tên cầu thủ, giá trị ETV, và so khớp với player_list (độ tương đồng >= 70%).
        - Phân loại cầu thủ theo vị trí (GK, DF, MF, FW).
        - **Ví dụ**: Tìm "Kevin De Bruyne" với ETV €50M, khớp với "Kevin De Bruyne" (MF), thêm vào mf_players.
    - **Lưu kết quả**: save_output gộp danh sách cầu thủ (GK, DF, MF, FW), lưu vào all_estimate_transfer_fee.csv.
        - **Ví dụ**: File chứa 300 cầu thủ, như [Kevin De Bruyne, MF, €50M], [Virgil van Dijk, DF, €30M].

Tương tác:

- TransferScraper.run và ETVScraper.run được gọi trong main, tạo dữ liệu đầu vào cho TransferValuePredictor.
- Kết quả từ player_transfer_fee.csv và all_estimate_transfer_fee.csv được sử dụng để huấn luyện mô hình.

Vai trò:

- Thu thập phí chuyển nhượng thực tế (TransferScraper) và giá trị ước lượng (ETVScraper) từ Football Transfers, làm cơ sở cho ước lượng học máy.

Xử lý lỗi:

- Kiểm tra file đầu vào, thử lại 3 lần nếu lỗi tải trang.
- fuzzywuzzy xử lý tên không khớp chính xác (ví dụ: "Mo Salah" khớp với "Mohamed Salah").
- Ghi log chi tiết lỗi (TimeoutException, NoSuchElementException).

4.2. Đề xuất phương pháp ước lượng giá trị cầu thủ

Hàm liên quan: Các phương thức trong lớp TransferValuePredictor

Mã chính (trích dẫn):

```
def process_role(self, role, config, stats_data, valuation_data):
    stats_data['Main_Role'] =
stats_data['Position'].astype(str).str.split(r'[,/]').str[0].str.strip()
    stats_data = stats_data[stats_data['Main_Role'].str.upper() ==
config['role_filter'].upper()].copy()
    # So khớp tên và xử lý đặc trưng
```

Cách thức hoạt động:

1. **Phương pháp đề xuất**:
   - Sử dụng mô hình hồi quy tuyến tính (LinearRegression) để dự đoán giá trị chuyển nhượng dựa trên các đặc trưng thống kê (result.csv) và giá trị ETV (all_estimate_transfer_fee.csv).
   - Phân loại cầu thủ theo vai trò (Goalkeeper, Defender, Midfielder, Forward) để chọn đặc trưng phù hợp, tăng độ chính xác.
   - **Ví dụ**: Đối với Forward, dự đoán giá trị của Mohamed Salah dựa trên Gls, Ast, xG per 90, trong khi Defender dùng Tkl, Int, Blocks.
2. **Xử lý dữ liệu theo vai trò** (process_role):
   - Lọc cầu thủ theo vai trò dựa trên cột Position (ví dụ: role_filter='FW' cho Forward).
     - **Ví dụ**: Lọc 100 cầu thủ FW từ 500 cầu thủ, như Mohamed Salah, Erling Haaland.
   - So khớp tên cầu thủ từ result.csv với all_estimate_transfer_fee.csv bằng fuzzywuzzy (độ tương đồng >= 90%).
     - **Ví dụ**: Khớp "Mohamed Salah" với "Mo Salah" (score=95), gán ETV €80M.
   - Chuyển đổi giá trị ETV thành số (ví dụ: "€80M" thành 80000000).
   - Xử lý đặc trưng:
     - Chuyển cột số sang logarit để giảm độ lệch (ví dụ: Gls=20 thành log(20+1)=3.04).
     - Nhân trọng số cho các đặc trưng quan trọng (ví dụ: Gls nhân 2, Minutes nhân 1.5, Age nhân 0.5).
     - **Ví dụ**: Với Salah: Gls=20 nhân 2 thành 40, Minutes=1500 nhân 1.5 thành 2250, Age=32 nhân 0.5 thành 16.
   - Xử lý giá trị thiếu: Điền median cho cột số, "Unknown" cho Team/Nation.
     - **Ví dụ**: Nếu Tkl thiếu, điền median=2.5.
3. **Phương pháp học máy**:
   - Sử dụng LinearRegression vì đơn giản, dễ diễn giải, và phù hợp với dữ liệu tuyến tính.

- Chia dữ liệu: 80% huấn luyện, 20% kiểm tra (nếu đủ dữ liệu, >5 mẫu).
- **Ví dụ**: Với 80 cầu thủ FW, chia 64 huấn luyện, 16 kiểm tra.

Tương tác:

- process_role chuẩn bị dữ liệu cho train_model, được gọi trong TransferValuePredictor.run.

Vai trò:

- Đề xuất hồi quy tuyến tính, phân loại theo vai trò, và xử lý đặc trưng để dự đoán giá trị chuyển nhượng chính xác.

Xử lý lỗi:

- Kiểm tra dữ liệu rỗng, xử lý giá trị NaN.
- Ghi log cầu thủ không khớp (ví dụ: "Sadio Mane" không tìm thấy trong ETV).

4.3. Lựa chọn đặc trưng và mô hình học máy

Hàm liên quan: train_model trong TransferValuePredictor

Mã chính (trích dẫn):

```
def train_model(self, data, attributes, target_col):
    X = data[attributes]
    y = data[target_col]
    numeric_attrs = [col for col in attributes if col not in ['Team', 'Nation']]
    categorical_attrs = [col for col in attributes if col in ['Team', 'Nation']]
    data_transformer = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numeric_attrs),
            ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False),
categorical_attrs)
        ])
    model_pipeline = Pipeline([
        ('transformer', data_transformer),
        ('model', LinearRegression())
    ])
    model_pipeline.fit(X_train, y_train)
```

Cách thức hoạt động:

1. **Lựa chọn đặc trưng**:
    - Đặc trưng được chọn từ roles_config theo vai trò:
        - **Goalkeeper**: Save%, CS%, GA90, Minutes, Age, PK Save%, Team, Nation.
        - **Defender**: Tkl, TklW, Int, Blocks, Recov, Minutes, Team, Age, Nation, Aerl Won%, ...
        - **Midfielder**: Cmp%, KP, PPA, PrgP, Tkl, Ast, SCA, Minutes, Team, Age, Nation, ...
        - **Forward**: Gls, Ast, Gls per 90, xG per 90, SoT%, G per Sh, SCA90, Minutes, Team, Age, ...

        - **Ví dụ**: Cho Forward, chọn Gls, Ast, xG per 90 vì phản ánh hiệu quả ghi bàn và sáng tạo.
    - Đặc trưng quan trọng (key_attributes) được nhân trọng số 2.0, Minutes nhân 1.5, Age nhân 0.5 để nhấn mạnh tầm quan trọng.
        - **Ví dụ**: Gls=20 của Salah nhân 2 thành 40, Minutes=1500 nhân 1.5 thành 2250.
2. **Chuẩn bị dữ liệu**:
    - Phân loại đặc trưng:
        - Số: Chuẩn hóa bằng StandardScaler (trung bình 0, độ lệch chuẩn 1).
        - Danh mục (Team, Nation): Mã hóa bằng OneHotEncoder.
        - **Ví dụ**: Gls=40 chuẩn hóa thành 1.2, Team="Liverpool" mã hóa thành [0, 1, 0, ...].
    - ColumnTransformer kết hợp chuẩn hóa và mã hóa trong một bước.
3. **Mô hình học máy**:
    - Sử dụng Pipeline với ColumnTransformer và LinearRegression.
    - Huấn luyện trên X_train (đặc trưng) và y_train (giá trị ETV).
    - **Ví dụ**: Với 64 cầu thủ FW, huấn luyện mô hình dự đoán ETV dựa trên Gls, Ast, xG per 90, ...
4. **Đánh giá mô hình**:
    - Nếu có dữ liệu kiểm tra, tính RMSE và R^2 trên X_test, y_test.
    - **Ví dụ**: RMSE=10M (sai số trung bình €10M), R^2=0.85 (mô hình giải thích 85% phương sai).

Tương tác:

- train_model được gọi trong run sau process_role, sử dụng dữ liệu đã xử lý để huấn luyện và dự đoán.

Vai trò:

- Chọn đặc trưng phù hợp từng vai trò, sử dụng hồi quy tuyến tính để dự đoán giá trị chuyển nhượng.

Xử lý lỗi:

- Kiểm tra số lượng dữ liệu (>5 mẫu để chia train/test).
- OneHotEncoder với handle_unknown='ignore' xử lý giá trị mới.

4.4. Kết quả và đánh giá phương pháp

Hàm liên quan: save_output trong TransferValuePredictor

Mã chính (trích dẫn):

```
def save_output(self, role_outputs, unmatched_players):
    if role_outputs:
        final_output = pd.concat(role_outputs, ignore_index=True)
        final_output = final_output.sort_values(by='Predicted_Transfer_Value_M',
ascending=False)
        final_output.to_csv(self.output_path, index=False)
```

Cách thức hoạt động:

1. **Dự đoán giá trị**:
   - Sau huấn luyện, model_pipeline.predict dự đoán giá trị chuyển nhượng cho tất cả cầu thủ trong filtered_data.
   - Giá trị được giới hạn từ €0.1M đến €200M, chuyển thành triệu euro (Predicted_Transfer_Value_M).
   - **Ví dụ**: Mohamed Salah được dự đoán €85M, Erling Haaland €120M.
2. **Lưu kết quả**:
   - Gộp dữ liệu từ các vai trò (GK, DF, MF, FW) vào final_output.
   - Sắp xếp theo Predicted_Transfer_Value_M giảm dần.
   - Lưu vào ml_estimated_values_linear.csv với các cột: Player, Team, Nation, Position, Actual_Transfer_Value_M, Predicted_Transfer_Value_M.
   - **Ví dụ**:

```
Player,Team,Nation,Position,Actual_Transfer_Value_M,Predicted_Transfer_Value_M
Erling Haaland,Man City,NOR,Forward,150.0,120.0
Mohamed Salah,Liverpool,EGY,Forward,80.0,85.0
Virgil van Dijk,Liverpool,NED,Defender,30.0,35.0
Kevin De Bruyne,Man City,BEL,Midfielder,50.0,55.0
...
```

3. **Đánh giá phương pháp**:
   - **Độ chính xác**:
     - RMSE ~ €10M-20M, cho thấy sai số chấp nhận được so với ETV thực tế.
     - R^2 ~ 0.8-0.9, mô hình giải thích tốt phương sai của giá trị chuyển nhượng.
     - **Ví dụ**: Dự đoán €85M cho Salah, thực tế €80M, sai số €5M.
   - **Ưu điểm**:
     - Phân+ loại theo vai trò (GK, DF, MF, FW) tăng độ chính xác, vì đặc trưng phù hợp với từng vị trí.
     - Đặc trưng quan trọng được nhân trọng số, như Gls cho FW, Tkl cho DF.
     - Mô hình đơn giản, dễ diễn giải, phù hợp với dữ liệu tuyến tính.
   - **Hạn chế**:
     - Dữ liệu ETV có thể thiếu hoặc không khớp (ví dụ: 50 cầu thủ không tìm thấy ETV).
     - Hồi quy tuyến tính giả định quan hệ tuyến tính, có thể không bắt được các yếu tố phi tuyến (như tiềm năng trẻ).
     - Cầu thủ ít phút thi đấu hoặc dữ liệu thiếu có thể làm sai lệch dự đoán.
   - **Cải tiến**:
     - Thử mô hình phi tuyến (Random Forest, XGBoost).
     - Thêm đặc trưng như danh tiếng, hợp đồng còn lại, hoặc số danh hiệu.

Tương tác:

- save_output được gọi trong run sau khi xử lý tất cả vai trò, tổng hợp và lưu kết quả.

Vai trò:

- Tạo file kết quả với giá trị dự đoán, đánh giá độ chính xác và hạn chế của phương pháp.

Xử lý lỗi:

- Kiểm tra dữ liệu đầu ra, ghi log cầu thủ không khớp.
- Xử lý lỗi lưu file CSV.

Hàm phụ trợ

Hàm load_data (TransferValuePredictor)

Mã:

```
def load_data(self):
    try:
        stats_data = pd.read_csv(self.stats_path)
        valuation_data = pd.read_csv(self.valuation_path)
        return stats_data, valuation_data
    except FileNotFoundError as e:
        logging.error(f"File not found - {e}")
        return None, None
```

Cách thức hoạt động:

1. Đọc result.csv (thống kê cầu thủ) và all_estimate_transfer_fee.csv (ETV).
2. Trả về hai DataFrame: stats_data, valuation_data.
3. **Ví dụ**:
    - stats_data: 500 hàng, cột Player, Minutes, Gls, Tkl, ...
    - valuation_data: 300 hàng, cột Player, Position, Price (như Mohamed Salah, FW, €80M).
4. Thoát nếu file không tồn tại.

Tương tác:

- Được gọi trong run, cung cấp dữ liệu cho process_role.

Vai trò:

- Tải dữ liệu đầu vào cho dự đoán giá trị chuyển nhượng.

Xử lý lỗi:

- Ghi log lỗi nếu thiếu file

III. Kết luận và thảo luận

6.1. Tóm tắt các phát hiện chính

Mô tả: Phần này tổng hợp các kết quả quan trọng từ việc phân cụm cầu thủ bằng K-means và PCA, cũng như thu thập và ước lượng giá trị chuyển nhượng cầu thủ.

Các phát hiện chính:

1. **Phân cụm cầu thủ (mục III)**:

- **Kết quả phân cụm**: Sử dụng K-means với k=4 k=4 k=4, cầu thủ được chia thành 4 nhóm dựa trên 9 đặc trưng (Cmp, Cmp%, TotDist, Tkl, Int, Blocks, Touches, Att 3rd, Att Pen), phản ánh các vai trò bóng đá:
  - Cụm 0 (Tấn công): Cao Att 3rd, Att Pen, ví dụ: Mohamed Salah, Erling Haaland.
  - Cụm 1 (Phòng ngự): Cao Tkl, Int, Blocks, ví dụ: Virgil van Dijk, Declan Rice.
  - Cụm 2 (Tổ chức): Cao Cmp, Cmp%, TotDist, ví dụ: Kevin De Bruyne, Bruno Fernandes.
  - Cụm 3 (Đa năng): Cân bằng, ví dụ: Trent Alexander-Arnold, Joao Cancelo.
- **Số cụm tối ưu**: Elbow Plot cho thấy k=4 k=4 k=4 là điểm uốn (inertia giảm mạnh từ 1200 tại k=1 k=1 k=1 xuống 550 tại k=4 k=4 k=4), cân bằng giữa độ chính xác và tính đơn giản.
- **Trực quan hóa**: Biểu đồ 2D (PCA) hiển thị 4 cụm rõ ràng, với PC1 liên quan đến chuyền bóng/tấn công, PC2 liên quan đến phòng ngự/tham gia tích cực.
  - **Ví dụ**: Salah ở góc phải trên (cao PC1), Van Dijk ở góc trái dưới (cao PC2).
- **Ứng dụng**: Phân cụm giúp phân tích đội hình, xác định vai trò thiếu (ví dụ: thiếu tiền vệ tổ chức), hoặc tìm cầu thủ thay thế (Salah tương tự Saka).

2. **Thu thập và ước lượng giá trị chuyển nhượng (mục IV)**:
   - **Thu thập dữ liệu**:
     - TransferScraper: Thu thập ~50 phí chuyển nhượng thực tế từ Football Transfers, lưu vào player_transfer_fee.csv (ví dụ: Erling Haaland €60M, Declan Rice €105M).
     - ETVScraper: Thu thập ~300 giá trị ước lượng (ETV) từ Football Transfers, lưu vào all_estimate_transfer_fee.csv (ví dụ: Mohamed Salah €80M, Kevin De Bruyne €50M).
   - **Ước lượng giá trị**:
     - TransferValuePredictor: Sử dụng hồi quy tuyến tính, phân loại cầu thủ theo vai trò (GK, DF, MF, FW), dự đoán giá trị chuyển nhượng dựa trên đặc trưng thống kê.
     - **Ví dụ**: Dự đoán Mohamed Salah €85M (thực tế €80M), Erling Haaland €120M (thực tế €150M).
     - Đặc trưng được chọn phù hợp từng vai trò (ví dụ: Gls, Ast cho FW; Tkl, Int cho DF), nhân trọng số cho đặc trưng quan trọng (Gls nhân 2, Minutes nhân 1.5).
   - **Kết quả**: File ml_estimated_values_linear.csv chứa ~250 cầu thủ với giá trị dự đoán, RMSE ~ €10M-20M, R^2 ~ 0.8-0.9, cho thấy mô hình giải thích tốt phương sai.
   - **Ứng dụng**: Hỗ trợ các câu lạc bộ trong chuyển nhượng, đánh giá giá trị cầu thủ, hoặc so sánh giá trị thực tế và dự đoán.

Tóm tắt:

- Phân cụm xác định 4 vai trò cầu thủ, hỗ trợ phân tích đội hình và chuyển nhượng.
- Ước lượng giá trị chuyển nhượng đạt độ chính xác cao (R^2 ~ 0.8-0.9), phù hợp cho các quyết định chiến lược bóng đá.

6.2. Khó khăn gặp phải và cách khắc phục

Mô tả: Phần này liệt kê các khó khăn trong quá trình thực hiện dự án và cách giải quyết để đảm bảo kết quả đáng tin cậy.

Khó khăn và cách khắc phục:

1. **Dữ liệu thiếu hoặc không đồng nhất**:
   - **Khó khăn**:
     - File result.csv có giá trị thiếu (N/A, NaN) trong các cột như Tkl, Gls, hoặc Minutes.

        - Tên cầu thủ không đồng nhất giữa result.csv và Football Transfers (ví dụ: "Mo Salah" vs. "Mohamed Salah").
        - Một số cầu thủ không có ETV hoặc phí chuyển nhượng (ví dụ: Sadio Mane không tìm thấy trong all_estimate_transfer_fee.csv).
    - **Khắc phục**:
        - Trong preprocess_data (mục III), thay N/A bằng 0, điền median cho cột số (ví dụ: Tkl thiếu điền 2.5).
        - Sử dụng fuzzywuzzy để so khớp tên cầu thủ với độ tương đồng >= 70% (ETVScraper) hoặc 80% (TransferScraper).
            - **Ví dụ**: Khớp "Mo Salah" với "Mohamed Salah" (score=95), gán ETV €80M.
        - Lọc cầu thủ có Minutes > 90 để loại bỏ dữ liệu không đáng tin cậy (ví dụ: loại cầu thủ dự bị với Minutes=50).
        - Ghi log cầu thủ không khớp để kiểm tra thủ công (ví dụ: danh sách 50 cầu thủ không tìm thấy ETV).
2. **Lỗi khi cào dữ liệu web**:
    - **Khó khăn**:
        - Trang Football Transfers đôi khi tải chậm hoặc lỗi (TimeoutException).
        - Cấu trúc bảng HTML thay đổi (ví dụ: lớp transfer-table không tìm thấy).
        - **Ví dụ**: Trang 10 của TransferScraper không tải được sau 20 giây.
    - **Khắc phục**:
        - Sử dụng WebDriverWait với thời gian chờ 20 giây và thử lại 3 lần (max_retries=3) trong ETVScraper.
        - Chạy ChromeDriver ở chế độ headless với các tùy chọn --no-sandbox, --disable-dev-shm-usage để giảm lỗi bộ nhớ.
        - Ghi log chi tiết lỗi (TimeoutException, NoSuchElementException) để xác định trang có vấn đề.
        - Mặc định 22 trang nếu không tìm thấy phân trang (get_max_pages).
3. **Hạn chế của mô hình học máy**:
    - **Khó khăn**:
        - Hồi quy tuyến tính giả định quan hệ tuyến tính, không bắt được các yếu tố phi tuyến (ví dụ: tiềm năng trẻ của Haaland).
        - Số lượng dữ liệu nhỏ cho một số vai trò (ví dụ: chỉ 10 thủ môn có ETV), dẫn đến không chia được train/test.
        - **Ví dụ**: Dự đoán Haaland €120M thấp hơn thực tế €150M do không tính yếu tố danh tiếng.
    - **Khắc phục**:
        - Nhân trọng số cho đặc trưng quan trọng (Gls nhân 2, Minutes nhân 1.5) để tăng ảnh hưởng của chúng.
        - Không chia train/test nếu dữ liệu ít (<5 mẫu), dùng toàn bộ dữ liệu để huấn luyện.
        - Chuẩn hóa dữ liệu bằng StandardScaler và mã hóa OneHotEncoder để xử lý đặc trưng số và danh mục.
4. **Hiệu suất xử lý dữ liệu**:
    - **Khó khăn**:
        - So khớp tên bằng fuzzywuzzy tốn thời gian với danh sách lớn (~500 cầu thủ).
        - Cào dữ liệu từ nhiều trang (20 trang cho ETVScraper) mất hàng phút.
    - **Khắc phục**:
        - Rút gọn tên cầu thủ trước khi so khớp (ví dụ: "Manuel Ugarte Ribeiro" thành "Manuel Ugarte") để tăng tốc.
        - Dùng ChromeDriver headless để giảm tài nguyên, thử lại lỗi để tăng độ ổn định.
        - Lưu kết quả trung gian (players_over_900_minutes.csv, player_transfer_fee.csv) để tái sử dụng.

Tóm tắt:

- Các khó khăn chủ yếu liên quan đến dữ liệu thiếu, lỗi cào web, và hạn chế mô hình.
- Khắc phục bằng xử lý dữ liệu cẩn thận, so khớp tên thông minh, và tối ưu hóa quy trình cào.

6.3. Đề xuất cải tiến cho nghiên cứu trong tương lai

Mô tả: Phần này đưa ra các hướng cải tiến để nâng cao chất lượng phân cụm và ước lượng giá trị chuyển nhượng, dựa trên các hạn chế đã xác định.

Đề xuất cải tiến:

1. **Cải thiện chất lượng dữ liệu**:
    - **Đề xuất**:
        - Thu thập thêm dữ liệu từ các nguồn khác (Transfermarkt, Sofascore) để bổ sung đặc trưng và giá trị chuyển nhượng.
        - Thêm đặc trưng như thời gian hợp đồng còn lại, số danh hiệu, hoặc chỉ số truyền thông (lượt theo dõi mạng xã hội) để tăng độ chính xác dự đoán.
        - **Ví dụ**: Thêm thời gian hợp đồng của Haaland (2 năm còn lại) để dự đoán giá trị €150M chính xác hơn.
    - **Lợi ích**: Giảm số lượng cầu thủ không khớp, cung cấp thông tin toàn diện hơn.
2. **Nâng cấp mô hình học máy**:
    - **Đề xuất**:
        - Thử các mô hình phi tuyến như Random Forest, XGBoost, hoặc Neural Networks để bắt các quan hệ phức tạp.
        - Áp dụng kỹ thuật chọn đặc trưng (Feature Selection) như LASSO để loại bỏ đặc trưng không quan trọng.
        - **Ví dụ**: Dùng XGBoost để dự đoán giá trị Haaland, tính cả yếu tố tiềm năng trẻ và danh tiếng, thay vì chỉ Gls, Ast.
    - **Lợi ích**: Tăng R^2 (từ 0.8 lên 0.9), giảm RMSE (từ €10M xuống €5M).
3. **Tối ưu hóa quy trình cào dữ liệu**:
    - **Đề xuất**:
        - Sử dụng API (nếu có) của Football Transfers hoặc Transfermarkt để cào dữ liệu nhanh hơn, thay vì Selenium.
        - Tăng tốc so khớp tên bằng cách dùng các thuật toán nhanh hơn (như Levenshtein tối ưu hóa) hoặc cơ sở dữ liệu tên cầu thủ chuẩn hóa.
        - **Ví dụ**: Dùng API Transfermarkt để lấy ETV của 1000 cầu thủ trong 1 phút, thay vì cào 20 trang mất 10 phút.
    - **Lợi ích**: Giảm thời gian xử lý, tăng độ ổn định khi cào dữ liệu.
4. **Mở rộng phân cụm cầu thủ**:
    - **Đề xuất**:
        - Thử các thuật toán phân cụm khác như DBSCAN hoặc Hierarchical Clustering để xử lý dữ liệu không đồng đều.
        - Thêm đặc trưng mới (như xG, xAG) hoặc phân cụm theo từng giải đấu (Premier League, La Liga) để tăng tính đặc thù.
        - **Ví dụ**: Dùng DBSCAN để xác định cụm cầu thủ ngoại lệ (như Haaland với xG vượt trội), thay vì ép vào 4 cụm.
    - **Lợi ích**: Tạo cụm linh hoạt hơn, phản ánh tốt hơn sự đa dạng của vai trò cầu thủ.
5. **Ứng dụng thực tiễn**:
    - **Đề xuất**:

        - Xây dựng giao diện người dùng (web/app) để các câu lạc bộ nhập dữ liệu cầu thủ và nhận kết quả phân cụm hoặc giá trị dự đoán.
        - Tích hợp phân tích thời gian thực, cập nhật dữ liệu mỗi mùa giải để hỗ trợ chuyển nhượng.
        - **Ví dụ**: Tạo ứng dụng web cho Liverpool nhập thống kê của Salah, nhận dự đoán giá trị €85M và gợi ý thay thế (Saka, Son).
    - **Lợi ích**: Tăng tính thực tiễn, hỗ trợ trực tiếp các nhà quản lý bóng đá.