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

o0o

BÁO CÁO BÀI TẬP LỚN 1

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn:	**Kim Ngọc Bách**
Sinh viên:	**Chu Tuyết Nhi**
Mã sinh viên:	**B23DCCE075**
Lớp:	**D23CQCEO6-B**
Niên khóa:	**2023 - 2028**
Hệ đào tạo:	**Đại học chính quy**

Hà Nội, 2025

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

o0o

BÁO CÁO BÀI TẬP LỚN 1

NGÔN NGỮ LẬP TRÌNH PYTHON

Giảng viên hướng dẫn:	**Kim Ngọc Bách**
Sinh viên:	**Chu Tuyết Nhi**
Mã sinh viên:	**B23DCCE075**
Lớp:	**D23CQCEO6-B**
Niên khóa:	**2023 - 2028**
Hệ đào tạo:	**Đại học chính quy**

Hà Nội, 2025

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20...

Giảng viên

Mục lục

Danh sách hình vẽ

Phần mở đầu

Báo cáo này được thực hiện nhằm hoàn thành yêu cầu của bài tập lớn số 1 môn lập trình Python. Mục tiêu chính của bài tập là áp dụng các kỹ thuật thu thập, phân tích và mô hình hóa dữ liệu để phân tích thông tin về các cầu thủ bóng đá Ngoại hạng Anh mùa giải 2024-2025.

Dữ liệu thống kê hiệu suất thi đấu của các cầu thủ đã chơi trên 90 phút được thu thập từ trang web fbref.com. Dữ liệu về giá trị chuyển nhượng ước tính của các cầu thủ thi đấu trên 900 phút được lấy từ footballtransfers.com. Tất cả quá trình thu thập dữ liệu này được thực hiện vào ngày 2 tháng 5 năm 2025. Do đó, mọi phân tích và kết quả trình bày trong báo cáo này phản ánh tình hình và số liệu của các cầu thủ tính đến thời điểm cụ thể đó của mùa giải. Nội dung báo cáo được chia thành bốn phần chính:

Chương 1. Thu thập dữ liệu cầu thủ từ fbref.com: Trình bày chi tiết quy trình thu thập dữ liệu thống kê cầu thủ từ fbref.com bằng cách sử dụng các công cụ tự động như Selenium và BeautifulSoup, cùng với các bước làm sạch và lưu trữ dữ liệu ban đầu. Làm theo yêu cầu Chương 1 của đề bài.

Chương 2. Phân tích và Trực quan hóa Dữ liệu: Thực hiện các phân tích thống kê mô tả, bao gồm xác định top 3 cầu thủ cao nhất/thấp nhất cho mỗi chỉ số, tính toán các giá trị trung bình, trung vị, độ lệch chuẩn, trực quan hóa phân phối dữ liệu qua biểu đồ histogram, và đánh giá sơ bộ hiệu suất các đội bóng.

Chương 3. Phân cụm Cầu thủ bằng K-Means và PCA: Áp dụng thuật toán phân cụm K-Means để nhóm các cầu thủ thành các nhóm có đặc điểm thống kê tương đồng, từ đó tìm hiểu các kiểu mẫu cầu thủ khác nhau trong giải đấu.

Chương 4. Ước tính giá trị cầu thủ. Phần này tập trung vào việc thu thập dữ liệu giá trị chuyển nhượng của các cầu thủ từ trang footballtransfers.com và đề xuất phương pháp xây dựng mô hình học máy (sử dụng Gradient Boosting Regressor) để ước tính giá trị thị trường của họ. Quá trình này bao gồm việc lựa chọn các đặc trưng từ dữ liệu thống kê hiệu suất (thu thập ở Chương 1), thông tin cơ bản của cầu thủ và giá trị chuyển nhượng lịch sử, sau đó huấn luyện và đánh giá mô hình để dự đoán giá trị cầu thủ.

Chương 1

Thu thập dữ liệu cầu thủ từ fbref.com

Phần này em trình bày phương pháp và quá trình thu thập dữ liệu thống kê cầu thủ giải Ngoại hạng Anh mùa 2024-2025 từ trang web fbref.com, thực hiện yêu cầu của Bài tập I. Dữ liệu cần thu thập bao gồm các thông số chi tiết đề bài Chương 1 đã yêu cầu cho các cầu thủ có thời gian thi đấu trên 90 phút. Kết quả được lưu vào tệp results.csv và sắp xếp theo tên cầu thủ.

1.1 Cấu trúc chương trình

Để thu thập dữ liệu thống kê cầu thủ từ trang fbref.com theo yêu cầu của đề bài, em đã trình bày một chương trình scraping được tổ chức theo hướng module hóa, bao gồm các thành phần chính sau:

`MAIN_part1.py`: Là module chính, chịu trách nhiệm điều phối toàn bộ quá trình thu thập dữ liệu.

`config_part1.py`: Lưu trữ các cấu hình quan trọng như URL của trang web, ánh xạ dữ liệu cần lấy, cũng như các tham số kỹ thuật như thời gian chờ.

`scraper.py`: Đảm nhiệm việc truy cập website và trích xuất dữ liệu thô từ các trang HTML.

`processor.py`: Xử lý dữ liệu vừa trích xuất, chuyển đổi chúng thành dạng DataFrame có cấu trúc rõ ràng để thuận tiện cho các bước phân tích sau.

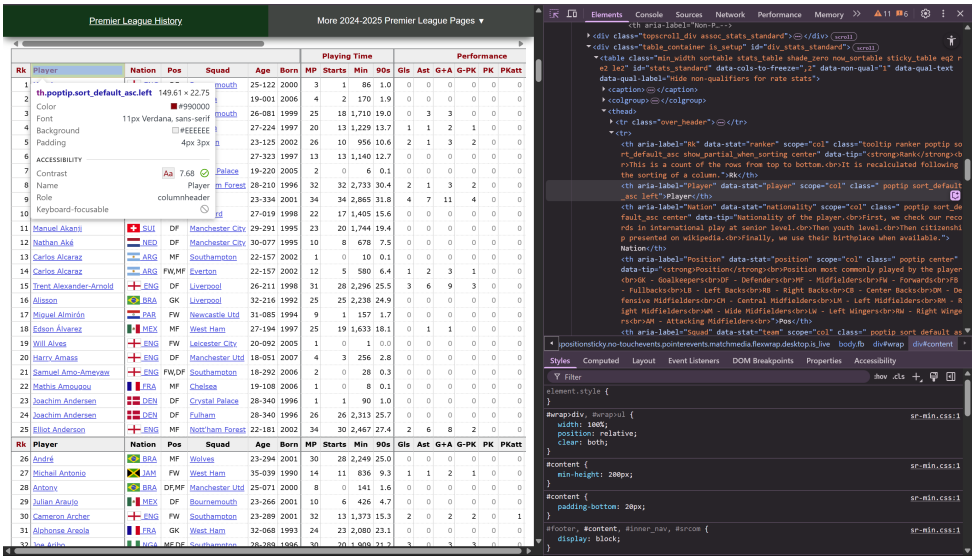
Việc thiết kế theo hướng module hóa giúp code có cấu trúc rõ ràng, dễ quản lý. Giúp em dễ dàng phát hiện và sửa lỗi trong quá trình thực hiện chương trình vì đã phân tách rõ ràng trách nhiệm của từng phần.

1.2 Thu thập dữ liệu với Selenium và BeautifulSoup

1.2.1 Lựa chọn công cụ thu thập dữ liệu

Em lựa chọn kết hợp giữa thư viện `Selenium` và `BeautifulSoup` do trang web fbref.com sử dụng JavaScript để tải dữ liệu động, điều mà thư viện `requests` không thể xử lý trực tiếp. Bên cạnh đó, `Selenium` cho phép mô phỏng hành vi người dùng một cách tự nhiên

hơn, từ đó giúp giảm thiểu nguy cơ bị chặn bởi các cơ chế chống thu thập dữ liệu tự động (anti-scraping) của trang web[1, 2].



Hình 1.1: Ảnh chụp màn hình Developer Tools đang highlight một ô dữ liệu trong bảng thống kê trên fbref.com và thẻ HTML tương ứng.

Ngoài ra, công cụ Developer Tools của trình duyệt cũng được sử dụng để xác định cấu trúc HTML của các bảng thống kê, xác định thuộc tính `data-stat` tương ứng với từng chỉ số cần thu thập.

1.2.2 Chi tiết thực hiện

Hàm chính `scrape_fbref_data` trong module `scraper.py` hoạt động theo các bước sau, với đoạn mã đã được rút gọn để tập trung vào logic chính:

```
def scrape_fbref_data(driver: WebDriver, url_config: dict, stats_map: dict) ->
    dict:
    player_data = {}
    for category, (url, table_id) in url_config.items():
            # 1. Truy cập URL của mỗi loại thống kê
            driver.get(url)
            # 2. Đợi cho bảng dữ liệu được tải hoàn chỉnh
            wait = WebDriverWait(driver, WAIT_TIME)
            wait.until(EC.presence_of_element_located((By.CSS_SELECTOR,
                                        f"{table_id} tbody tr")))
            # 3. Phân tích HTML với BeautifulSoup
            soup = BeautifulSoup(driver.page_source, 'html.parser')
            rows = soup.select(f"{table_id} tbody tr")
            # 4. Xử lý từng dòng dữ liệu cầu thủ
            for row in rows:
                if 'thead' in row.get('class', []): continue
```

```
                player_name = safe_get_text(row, 'player')
                team_name = safe_get_text(row, 'team')
                if not player_name or team_name == 'N/a': continue
                # 5. Tạo khóa duy nhất cho mỗi cầu thủ là cặp (tên, đội)
                key = (player_name, team_name)
                data = player_data.setdefault(key, {'Player': player_name,
                                                    'Team': team_name})
                # 6. Thu thập tất cả thống kê theo ánh xạ đã cấu hình
                for k, stat in stats_map.items():
                    if k not in ['Player', 'Team']:
                        val = safe_get_text(row, stat)
                        if val != 'N/a' or k not in data:
                            data[k] = val
    return player_data
```

Hàm `scrape_fbref_data` được thiết kế để tự động thu thập và trích xuất dữ liệu thống kê cầu thủ từ nhiều trang web khác nhau trên trang FBref. Hàm nhận vào một trình điều khiển `Selenium (driver)`, một cấu hình các URL và ID bảng (`url_config`), cùng với một ánh xạ tên cột (stats_map). Quá trình thực hiện bao gồm: truy cập từng URL, đợi bảng dữ liệu tải xong, phân tích HTML bằng BeautifulSoup, rồi trích xuất và tổng hợp dữ liệu của từng cầu thủ thành một cấu trúc từ điển với khóa là cặp (tên cầu thủ, tên đội bóng). Hàm đảm bảo chỉ thu thập các dữ liệu hợp lệ và trả về kết quả dưới dạng một từ điển, tiện cho việc xử lý và lưu trữ sau này.

Do tình trạng chuyển nhượng diễn ra trong mùa giải, một số cầu thủ có thể thi đấu cho nhiều câu lạc bộ khác nhau. Để đảm bảo tính chính xác và toàn vẹn của dữ liệu, em lựa chọn phương pháp phân tách dữ liệu theo từng đội bóng mà cầu thủ đã thi đấu. Cụ thể, trong quá trình thu thập dữ liệu từ trang fbref.com thông qua đoạn mã trong tệp `scraper.py`, mỗi cầu thủ được định danh duy nhất bằng cặp thông tin (Tên cầu thủ, Tên đội bóng). Cách tiếp cận này mang lại một số lợi ích như sau:

- **Độ chính xác:** Hiệu suất thi đấu của cầu thủ được thống kê riêng biệt theo từng câu lạc bộ, phản ánh đúng đóng góp của họ trong từng giai đoạn cụ thể.
- **Tính toàn vẹn:** Không bỏ sót dữ liệu khi cầu thủ thay đổi đội bóng trong mùa giải. Mỗi giai đoạn tham gia với một đội bóng đều được ghi nhận riêng biệt.
- **Tương thích với dữ liệu nguồn:** fbref.com cũng trình bày dữ liệu theo phương pháp này đối với các cầu thủ thi đấu cho nhiều câu lạc bộ trong cùng một mùa giải.

1.3 Cấu hình và ánh xạ dữ liệu

Để dễ dàng thay đổi các thông số như URL hay tên trường cần lấy, em đã đưa hết chúng vào file `config_part1.py` thay vì viết cứng trong code. Em dùng hai cấu trúc map chính để quản lý các thông tin này:

`URL_CONFIG`: Ánh xạ giữa loại thống kê và URL tương ứng:

```
URL_CONFIG = {
    'standard': ('https://fbref.com/en/comps/9/stats/Premier-League-Stats',
    ↪ '#stats_standard'),
    'keeper': ('https://fbref.com/en/comps/9/keepers/Premier-League-Stats',
    ↪ '#stats_keeper'),
    'shooting': ('https://fbref.com/en/comps/9/shooting/Premier-League-Stats',
    ↪ '#stats_shooting'),
    # ... các loại thống kê khác
}
```

`STATS_MAP`: Ánh xạ giữa tên cột trong CSV với thuộc tính `data-stat` trong HTML:

```
STATS_MAP = {
    'Player': 'player',
    'Nation': 'nationality',
    'Team': 'team',
    # ... các chỉ số thống kê khác
}
```

1.4 Xử lý và làm sạch dữ liệu

Module `processor.py` chịu trách nhiệm chuyển đổi dữ liệu thô thành DataFrame có cấu trúc phù hợp với yêu cầu, với đoạn mã đã được rút gọn để tập trung vào logic chính và các bước code thực hiện:

```
def process_data(raw_data: dict, final_columns: list) -> pd.DataFrame:
    # 1. Chuyển đổi dữ liệu từ dict sang DataFrame
    df = pd.DataFrame.from_dict(raw_data, orient='index')
    # 2. Lọc cầu thủ chơi hơn 90 phút
    minutes_column_name = 'Playing Time: minutes' # Cần đảm bảo tên cột chính
    ↪ xác
    if minutes_column_name in df.columns:
        df['Min_numeric'] = pd.to_numeric(
            df[minutes_column_name].astype(str).str.replace(',', '',
            ↪ regex=False), errors='coerce')
        df = df[df['Min_numeric'] > 90].copy()
        df.drop(columns=['Min_numeric'], inplace=True)
    # 3. Chọn các cột theo thứ tự đã xác định
    # Đảm bảo final_columns chứa các cột tồn tại trong df sau khi lọc
    existing_columns = [col for col in final_columns if col in df.columns]
    df = df[existing_columns]
    # 4. Điền 'N/a' cho dữ liệu thiếu
```

```
    df.fillna('N/a', inplace=True)
    # 5. Sắp xếp theo tên cầu thủ
    if 'Player' in df.columns:
         df.sort_values(
             by='Player',
             key=lambda x: x.str.split().str[0].str.lower(),
             inplace=True
         )
    return df
```

Các xử lý chính bao gồm:

- **Lọc dữ liệu cầu thủ:** Chỉ giữ lại những cầu thủ có tổng thời gian thi đấu lớn hơn 90 phút. Do dữ liệu thời gian trên trang web được định dạng với dấu phẩy để ngăn cách các đơn vị (ví dụ: "1,234" phút), cần thực hiện bước xử lý để loại bỏ dấu phẩy trước khi chuyển đổi sang kiểu dữ liệu số.
- **Xử lý dữ liệu thiếu:** Điền 'N/a' cho tất cả giá trị NaN theo yêu cầu đề bài.
- **Sắp xếp dữ liệu:** Sắp xếp cầu thủ theo tên (đặc biệt là tên đầu tiên, không phải họ).

1.5 Hàm thực hiện chính

Module chính `MAIN_part1.py` quản lý quá trình thực thi hoàn chỉnh, đoạn mã đã được rút gọn để tập trung vào logic chính:

```
def run_scraper():
    # Bước 1: Khởi tạo WebDriver
    driver = webdriver.Chrome(options=Options().add_argument('--log-level=3'))

    # Bước 2: Thu thập và xử lý dữ liệu
    raw_data = scrape_fbref_data(driver, URL_CONFIG, STATS_MAP) # từ
    ↪  scraper.py
    # Đưa dữ liệu vào hàm xử lý
    df = process_data(raw_data, list(STATS_MAP)) # từ processor.py

    # Bước 3: Lưu kết quả thành file csv
    df.to_csv(OUTPUT_FILENAME, index=False, encoding='utf-8-sig')

    # Đóng WebDriver
    driver.quit()
```

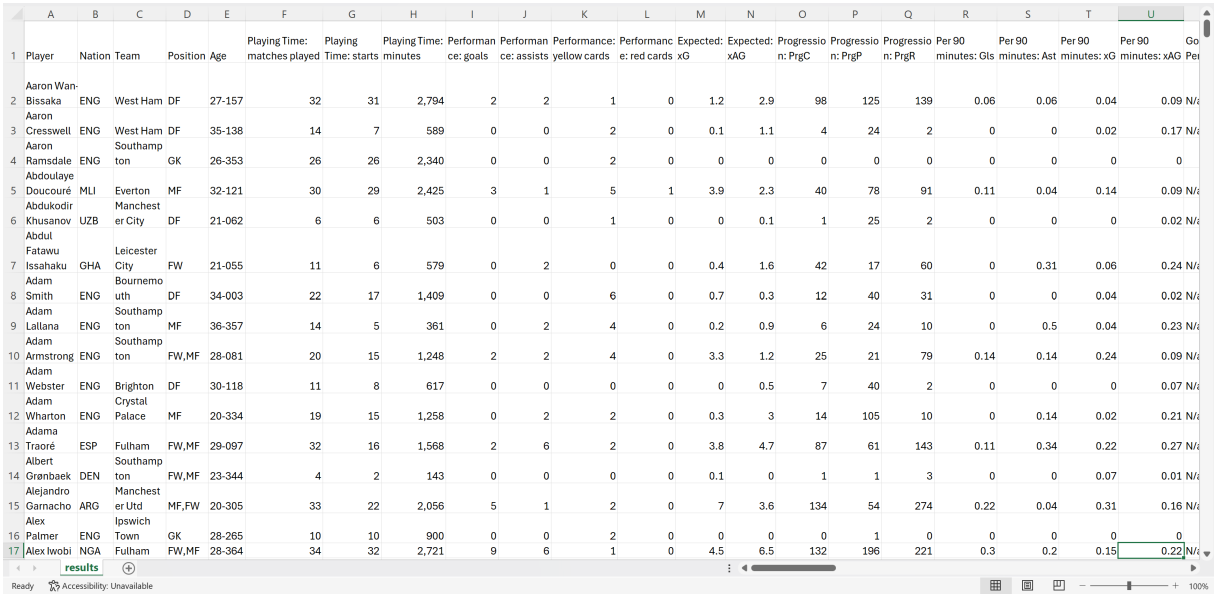
Đoạn mã trên minh họa cách thức hoạt động tổng thể của hàm `run_scraper`, bao gồm các bước: khởi tạo WebDriver, thu thập và xử lý dữ liệu, lưu kết quả đầu ra dưới dạng tệp CSV với mã hóa `utf-8-sig` để đảm bảo tính toàn vẹn dữ liệu, và cuối cùng là đóng WebDriver.

1.6 Kết quả thu thập dữ liệu

Kết quả cuối cùng được lưu trong file results.csv, và cụ thể:

- **Số lượng cầu thủ:** 491 cầu thủ (đã chơi trên 90 phút).
- **Số thống kê mỗi cầu thủ:** các chỉ số theo yêu cầu đề bài.
- **Định dạng dữ liệu:** CSV chuẩn với tiêu đề cột.
- **Sắp xếp:** Cầu thủ được sắp xếp theo tên đầu tiên.
- **Dữ liệu thiếu:** Đánh dấu là "N/a"theo đề yêu cầu.

Dưới đây là một đoạn mẫu từ file results.csv được mở trong Excel:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
1	Player	Nation	Team	Position	Age	Playing Time: matches played	Playing Time: starts	Playing Time: minutes	Performance: goals	Performance: assists	Performance: yellow cards	Performance: red cards	Expected: xG	Expected: xAG	Progression: PrgC	Progression: PrgP	Progression: PrgR	Per 90 minutes: Gls	Per 90 minutes: Ast	Per 90 minutes: xG	Per 90 minutes: xAG	Go Pe
2	Aaron Wan-Bissaka	ENG	West Ham	DF	27-157	32	31	2,794	2	2	1	0	1.2	2.9	98	125	139	0.06	0.06	0.04	0.09	N/a
3	Aaron Cresswell	ENG	West Ham	DF	35-138	14	7	589	0	0	2	0	0.1	1.1	4	24	2	0	0	0.02	0.17	N/a
4	Aaron Ramsdale	ENG	Southampton	GK	26-353	26	26	2,340	0	0	2	0	0	0	0	0	0	0	0	0	0	
5	Abdoulaye Doucouré	MLI	Everton	MF	32-121	30	29	2,425	3	1	5	1	3.9	2.3	40	78	91	0.11	0.04	0.14	0.09	N/a
6	Abdukodir Khusanov	UZB	Manchester City	DF	21-062	6	6	503	0	0	1	0	0	0.1	1	25	2	0	0	0	0.02	N/a
7	Abdul Fatawu Issahaku	GHA	Leicester City	FW	21-055	11	6	579	0	2	0	0	0.4	1.6	42	17	60	0	0.31	0.06	0.24	N/a
8	Adam Smith	ENG	Bournemouth	DF	34-003	22	17	1,409	0	0	6	0	0.7	0.3	12	40	31	0	0	0.04	0.02	N/a
9	Adam Lallana	ENG	Southampton	MF	36-357	14	5	361	0	2	4	0	0.2	0.9	6	24	10	0	0.5	0.04	0.23	N/a
10	Adam Armstrong	ENG	Southampton	FW,MF	28-081	20	15	1,248	2	2	4	0	3.3	1.2	25	21	79	0.14	0.14	0.24	0.09	N/a
11	Adam Webster	ENG	Brighton	DF	30-118	11	8	617	0	0	0	0	0	0.5	7	40	2	0	0	0	0.07	N/a
12	Adam Wharton	ENG	Crystal Palace	MF	20-334	19	15	1,258	0	2	2	0	0.3	3	14	105	10	0	0.14	0.02	0.21	N/a
13	Adama Traoré	ESP	Fulham	FW,MF	29-097	32	16	1,568	2	6	2	0	3.8	4.7	87	61	143	0.11	0.34	0.22	0.27	N/a
14	Albert Grønbaek	DEN	Southampton	FW,MF	23-344	4	2	143	0	0	0	0	0.1	0	1	1	3	0	0	0.07	0.01	N/a
15	Alejandro Garnacho	ARG	Manchester Utd	MF,FW	20-305	33	22	2,056	5	1	2	0	7	3.6	134	54	274	0.22	0.04	0.31	0.16	N/a
16	Alex Palmer	ENG	Ipswich Town	GK	28-265	10	10	900	0	0	2	0	0	0	0	1	0	0	0	0	0	
17	Alex Iwobi	NGA	Fulham	FW,MF	28-364	34	32	2,721	9	6	1	0	4.5	6.5	132	196	221	0.3	0.2	0.15	0.22	N/a

Hình 1.2: Đoạn mẫu từ file results.csv.

Chương 2

Phân tích và Trực quan hóa Dữ liệu

Phần này tập trung vào việc phân tích sâu hơn bộ dữ liệu cầu thủ đã thu thập ở Chương 1, thực hiện các yêu cầu của Bài tập II. Mục tiêu là phân tích các đặc điểm nổi bật của dữ liệu thông qua thống kê mô tả, xác định các cầu thủ và đội bóng có hiệu suất cao/thấp, và trực quan hóa sự phân phối của các chỉ số quan trọng qua biểu đồ. Đồng thời phân tích tìm đội bóng có hiệu suất cao nhất như đề bài yêu cầu.

2.1 Cấu trúc chương trình Chương 2

Tương tự như Chương 1, chương trình cho Chương 2 cũng được tổ chức theo hướng module hóa để đảm bảo tính rõ ràng và dễ bảo trì:

`MAIN_part2.py`: Module điều phối chính, thực hiện tuần tự các bước phân tích và gọi các hàm chức năng từ các module khác.

`config_part2.py`: Chứa các cấu hình cần thiết cho Chương 2, bao gồm đường dẫn đến tệp dữ liệu đầu vào (results.csv từ Chương 1), các thư mục đầu ra, danh sách các chỉ số thống kê cần phân tích (tấn công, phòng ngự, 3 chỉ số phòng thủ và 3 tấn công được chọn lọc, các chỉ số tiêu cực), và các cột cần loại trừ khỏi một số phân tích.

`analysis.py`: Module chứa các hàm thực hiện các phép phân tích cốt lõi như tìm top/bottom cầu thủ, tính toán thống kê tóm tắt (trung vị, trung bình, độ lệch chuẩn) theo đội và tổng thể, và tìm đội có hiệu suất tốt nhất của từng chỉ số.

`plotting.py`: Module chịu trách nhiệm tạo các biểu đồ trực quan hóa dữ liệu, cụ thể là biểu đồ histogram phân phối của các chỉ số được chọn lọc.

2.2 Chuẩn bị và tiền xử lý dữ liệu

Bước đầu tiên là tải dữ liệu từ tệp `results.csv` đã được tạo ở Chương 1.

```
# Trích đoạn từ MAIN_part2.py - Hàm load_data
def load_data():
    df = pd.read_csv(INPUT_CSV)  # đọc file CSV
    numeric_cols = [col for col in df.columns if col not in
    ↪  EXCLUDED_COLUMNS_FROM_TOP3]  # chọn cột số
```

```
    for col in numeric_cols:
        df[col] = pd.to_numeric(df[col].astype(str).str.replace(',', '',
        ↪  regex=False), errors='coerce')  # chuyển sang số
    return df, numeric_cols  # trả về dataframe và danh sách cột số
```

Trong quá trình này, các cột chứa dữ liệu thống kê (ngoại trừ các cột định danh như Tên, Quốc tịch, Đội bóng, Vị trí, Tuổi) được chuyển đổi sang định dạng số để có thể thực hiện các phép tính toán. Hàm `pd.to_numeric` được sử dụng với tham số `errors='coerce'` để tự động chuyển các giá trị không hợp lệ (ví dụ: 'N/a') thành NaN, đồng thời loại bỏ dấu phẩy ngăn cách hàng nghìn (nếu có).

2.3 Xác định Top 3 và Bottom 3 Cầu thủ theo từng chỉ số

Để xác định những cầu thủ nổi bật, cao nhất và thấp nhất cho mỗi chỉ số thống kê, hàm `get_top_bottom_players` trong `analysis.py` được sử dụng. Hàm này sắp xếp DataFrame dựa trên cột chỉ số được chỉ định và trả về N cầu thủ hàng đầu và N cầu thủ cuối cùng (N=3 theo yêu cầu).

```
# Trích đoạn từ analysis.py - Hàm get_top_bottom_players
def get_top_bottom_players(df, stat_col, n=3):
    # 1. lấy cột cần thiết
    df_valid = df[['Player', 'Team', stat_col]].copy()
    # 2. chuyển sang số
    df_valid[stat_col] = pd.to_numeric(df_valid[stat_col], errors='coerce')
    df_valid = df_valid.dropna(subset=[stat_col])  # 3. bỏ giá trị NaN
    # 4. sắp xếp giảm dần
    df_sorted = df_valid.sort_values(stat_col, ascending=False)
     # 5. top n và bottom n
    return df_sorted.head(n), df_sorted.tail(n).sort_values(stat_col)
```

Trong phần này, các giá trị 'N/a' được loại bỏ thay vì thay thế bằng các giá trị khác như 0 vì những lý do sau:

- 'N/a' không đồng nghĩa với giá trị 0. Việc thay thế bằng 0 gây hiểu sai, vì nó ngụ ý cầu thủ đã thi đấu nhưng không ghi bàn, trong khi thực tế có thể là dữ liệu bị thiếu. Điều này làm sai lệch bản chất dữ liệu và ảnh hưởng đến độ chính xác trong xếp hạng.
- Đảm bảo tính công bằng khi xếp hạng: Để xác định Top/Bottom 3 một cách công bằng và chính xác, chỉ nên so sánh những cầu thủ có đầy đủ dữ liệu hợp lệ cho chỉ số đó.

Kết quả đạt được: Dưới đây là ví dụ về top/bottom 3 cầu thủ cho 2 chỉ số tiêu biểu, trích từ tệp `top_3.txt`:

Nhận xét chung: Dữ liệu cho thấy sự khác biệt lớn về hiệu suất và thời gian thi đấu giữa các cầu thủ. Một số cầu thủ nổi bật ở nhiều hạng mục, trong khi những người khác có đóng góp hạn chế hơn hoặc chuyên biệt cho vai trò của họ.

Bảng 2.1: Playing Time: matches played

Cầu thủ	Đội	Số trận
Top 3		
Youri Tielemans	Aston Villa	34
Virgil van Dijk	Liverpool	34
Bruno Guimarães	Newcastle Utd	34
Bottom 3		
Jahmai Simpson-Pusey	Manchester City	2
Ayden Heaven	Manchester Utd	2
Hákon Rafn Valdimarsson	Brentford	2

Nhận xét: Ba cầu thủ dẫn đầu đều có số lần ra sân tối đa là 34 trận, cho thấy vai trò quan trọng và sự ổn định của họ trong đội hình. Ngược lại, nhóm cuối bảng có rất ít cơ hội thi đấu.

Bảng 2.2: Performance: goals

Cầu thủ	Đội	Bàn thắng
Top 3		
Mohamed Salah	Liverpool	28
Alexander Isak	Newcastle Utd	22
Erling Haaland	Manchester City	21
Bottom 3		
Chiedozie Ogbene	Ipswich Town	0
Cheick Doucouré	Crystal Palace	0
Adam Webster	Brighton	0

Nhận xét: Mohamed Salah của Liverpool dẫn đầu danh sách ghi bàn với 28 pha lập công, tạo khoảng cách đáng kể với các cầu thủ xếp sau. Nhiều cầu thủ, đặc biệt là những người có thiên hướng phòng ngự hoặc ít được ra sân, chưa ghi được bàn thắng nào.

2.4 Tính toán Thống kê Tóm tắt

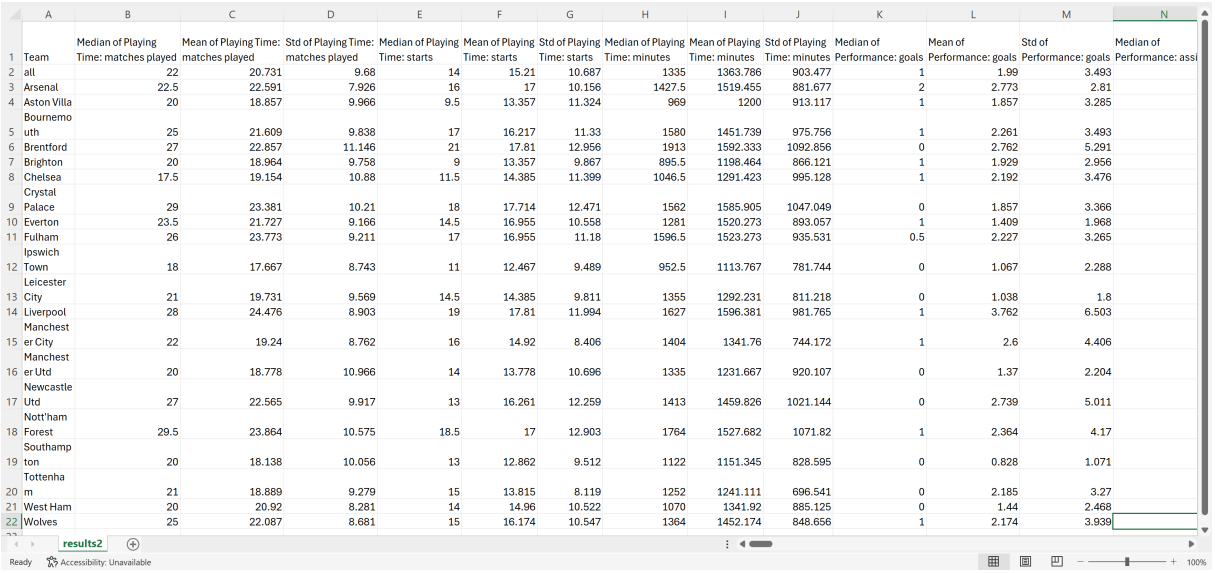
Bước tiếp theo là tính toán các đại lượng thống kê mô tả cơ bản (trung vị - median, trung bình - mean, độ lệch chuẩn - standard deviation) cho từng chỉ số. Các giá trị này được tính toán cho toàn bộ giải đấu ("all") và cho từng đội bóng riêng biệt. Hàm `calculate_stats_summary` trong `analysis.py` đảm nhận việc này.

```
# Trích đoạn từ analysis.py - Hàm calculate_stats_summary
def calculate_stats_summary(df, stats_cols):
    # ... (Xác thực và chuẩn bị cột dữ liệu số) ...
    team_stats = pd.DataFrame()
    if 'Team' in df_copy.columns and not df_copy['Team'].isnull().all():
```

```
        # Gom nhóm theo đội và tính toán median, mean, std cho các cột số liệu
        grouped = df_copy.groupby('Team')
        team_stats_agg = grouped[valid_cols].agg(['median', 'mean', 'std'])
        # ... (Xử lý tên cột sau khi aggregate) ...
        team_stats = team_stats_agg.reset_index()
        # ... (Đổi tên cột cho rõ ràng hơn) ...
    # ... (Xử lý trường hợp cột Team bị thiếu) ...
    # Tính toán thống kê tổng thể cho toàn bộ cầu thủ ("all")
    overall_stats_data = {'Team': 'all'}
    for col in valid_cols:
        overall_stats_data[f'Median of {col}'] = df_copy[col].median()
        overall_stats_data[f'Mean of {col}'] = df_copy[col].mean()
        overall_stats_data[f'Std of {col}'] = df_copy[col].std()
    overall_df = pd.DataFrame([overall_stats_data])
    # Kết hợp kết quả tổng thể và kết quả theo từng đội
    summary = pd.concat([overall_df_ordered, team_stats], ignore_index=True)
    return summary
```

Kết quả đạt được: Sau khi thực hiện xử lý và phân tích dữ liệu cho 20 đội bóng, cùng với một đội "All"đại diện cho tổng hợp toàn bộ các đội, kết quả thu được đã phản ánh khá đầy đủ và trực quan về các chỉ số chính liên quan đến hiệu suất thi đấu của từng đội. Việc tổng hợp thành đội "All"giúp tạo cái nhìn tổng quan, làm cơ sở đối chiếu với hiệu suất riêng lẻ của từng đội.

Dưới đây là một đoạn mẫu từ file results2.csv được mở trong Excel:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Team	Median of Playing Time: matches played	Mean of Playing Time: matches played	Std of Playing Time: matches played	Median of Playing Time: starts	Mean of Playing Time: starts	Std of Playing Time: starts	Median of Playing Time: minutes	Mean of Playing Time: minutes	Std of Playing Time: minutes	Median of Performance: goals	Mean of Performance: goals	Std of Performance: goals	Median of Performance: assi
2	all	22	20.731	9.68	14	15.21	10.687	1335	1363.786	903.477	1	1.99	3.493	
3	Arsenal	22.5	22.591	7.926	16	17	10.156	1427.5	1519.455	881.677	2	2.773	2.81	
4	Aston Villa	20	18.857	9.966	9.5	13.357	11.324	969	1200	913.117	1	1.857	3.285	
5	Bournemouth	25	21.609	9.838	17	16.217	11.33	1580	1451.739	975.756	1	2.261	3.493	
6	Brentford	27	22.857	11.146	21	17.81	12.956	1913	1592.333	1092.856	0	2.762	5.291	
7	Brighton	20	18.964	9.758	9	13.357	9.867	895.5	1198.464	866.121	1	1.929	2.956	
8	Chelsea	17.5	19.154	10.88	11.5	14.385	11.399	1046.5	1291.423	995.128	1	2.192	3.476	
9	Crystal Palace	29	23.381	10.21	18	17.714	12.471	1562	1585.905	1047.049	0	1.857	3.366	
10	Everton	23.5	21.727	9.166	14.5	16.955	10.558	1281	1520.273	893.057	1	1.409	1.968	
11	Fulham	26	23.773	9.211	17	16.955	11.18	1596.5	1523.273	935.531	0.5	2.227	3.265	
12	Ipswich Town	18	17.667	8.743	11	12.467	9.489	952.5	1113.767	781.744	0	1.067	2.288	
13	Leicester City	21	19.731	9.569	14.5	14.385	9.811	1355	1292.231	811.218	0	1.038	1.8	
14	Liverpool	28	24.476	8.903	19	17.81	11.994	1627	1596.381	981.765	1	3.762	6.503	
15	Manchester City	22	19.24	8.762	16	14.92	8.406	1404	1341.76	744.172	1	2.6	4.406	
16	Manchester Utd	20	18.778	10.966	14	13.778	10.696	1335	1231.667	920.107	0	1.37	2.204	
17	Newcastle Utd	27	22.565	9.917	13	16.261	12.259	1413	1459.826	1021.144	0	2.739	5.011	
18	Nott'ham Forest	29.5	23.864	10.575	18.5	17	12.903	1764	1527.682	1071.82	1	2.364	4.17	
19	Southampton	20	18.138	10.056	13	12.862	9.512	1122	1151.345	828.595	0	0.828	1.071	
20	Tottenham	21	18.889	9.279	15	13.815	8.119	1252	1241.111	696.541	0	2.185	3.27	
21	West Ham	20	20.92	8.281	14	14.96	10.522	1070	1341.92	885.125	0	1.44	2.468	
22	Wolves	25	22.087	8.681	15	16.174	10.547	1364	1452.174	848.656	1	2.174	3.939	

Hình 2.1: Đoạn mẫu từ file results2.csv.

2.5 Trực quan hóa Phân phối Dữ liệu

Để hiểu rõ hơn về sự phân phối của các chỉ số thống kê, theo yêu cầu đề bài biểu đồ histogram được sử dụng. Module `plotting.py` cung cấp các hàm để vẽ:

Histogram tổng thể (`plot_histogram_all_players`): Cho thấy phân phối của một chỉ số trên tất cả các cầu thủ trong giải đấu. Biểu đồ này đi kèm đường ước lượng mật độ (KDE) để làm rõ xu hướng phân phối.

```
def plot_histogram_all_players(df, stat_col, bins=20, fmt='png', xlim=None, ylim=None):
# Tạo figure và axes để vẽ biểu đồ
fig, ax = plt.subplots(figsize=(10, 6))
# Vẽ biểu đồ histogram có đường KDE cho toàn bộ người chơi
sns.histplot(df[stat_col].dropna(), bins=bins, kde=True, ax=ax)
# Đặt tiêu đề và nhãn trục
ax.set(title=f'Distribution of {stat_col} (All Players)', xlabel=stat_col, ylabel='Frequency')
# Giới hạn trục nếu có thiết lập
if xlim: ax.set_xlim(xlim)
if ylim: ax.set_ylim(ylim)
# Căn chỉnh layout và lưu hình
fig.tight_layout()
_save_plot(fig, f'hist_all_{_safe_filename(stat_col)}', fmt)
```

Histogram theo từng đội (`plot_histograms_per_team_facet`): Hiển thị nhiều biểu đồ histogram nhỏ , mỗi biểu đồ tương ứng với một đội, giúp so sánh trực quan sự khác biệt về phân phối chỉ số giữa các đội.

```
def plot_histograms_per_team_facet(df, stat_col, col_wrap=4, bins=15, fmt='png', xlim=None, ylim=None):
# Loại bỏ các dòng thiếu giá trị ở cột thống kê hoặc tên đội
df_valid = df.dropna(subset=[stat_col, 'Team'])
# Tạo lưới biểu đồ theo từng đội
g = sns.FacetGrid(df_valid, col="Team", col_wrap=col_wrap, sharex=True, sharey=False, height=3, aspect=1.2)
g.map(sns.histplot, stat_col, bins=bins)
# Áp dụng giới hạn trục nếu có
if xlim or ylim:
    for ax in g.axes.flatten():
        if xlim: ax.set_xlim(xlim)
        if ylim: ax.set_ylim(ylim)
# Đặt tiêu đề và nhãn trục
g.set_titles("{col_name}")
g.set_axis_labels(stat_col, "Frequency")
plt.suptitle(f'Distribution of {stat_col} per Team', y=1.02)
# Căn chỉnh layout và lưu hình
```

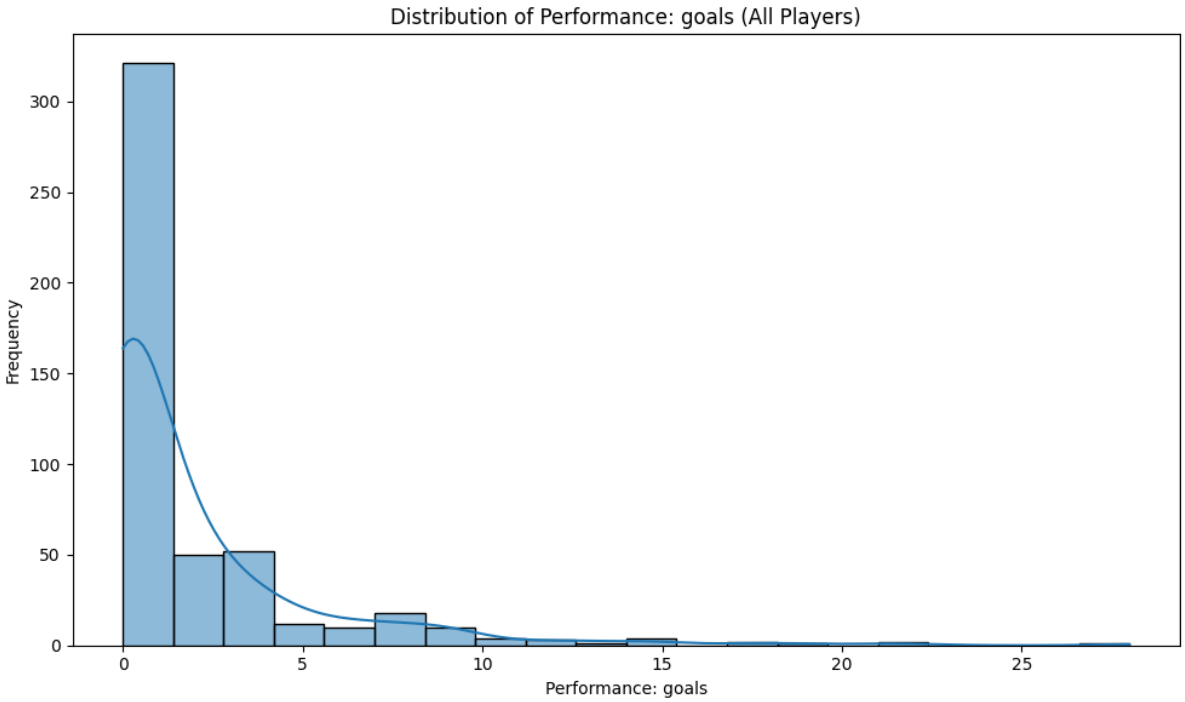
```
        g.tight_layout(rect=[0, 0.03, 1, 0.98])
        _save_plot(g.fig, f'hist_facet_team_{_safe_filename(stat_col)}', fmt)
```

Các biểu đồ này được tạo cho các 3 chỉ số tấn công và phòng ngự được chọn lọc trong `config_part2.py`, vì đây được coi là 6 chỉ số quan trọng và cân bằng với nhau trong bóng đá, bao gồm:

- Performance: goals (Bàn thắng)
- Performance: assists (Kiến tạo)
- Shooting: Standard: SoT/90 (Sút trúng đích mỗi 90 phút)
- Defensive Actions: Tackles: TklW (Tắc bóng thành công)
- Defensive Actions: Blocks: Int (Cắt bóng)
- Miscellaneous: Aerial Duels: Won% (Tỷ lệ không chiến thành công)

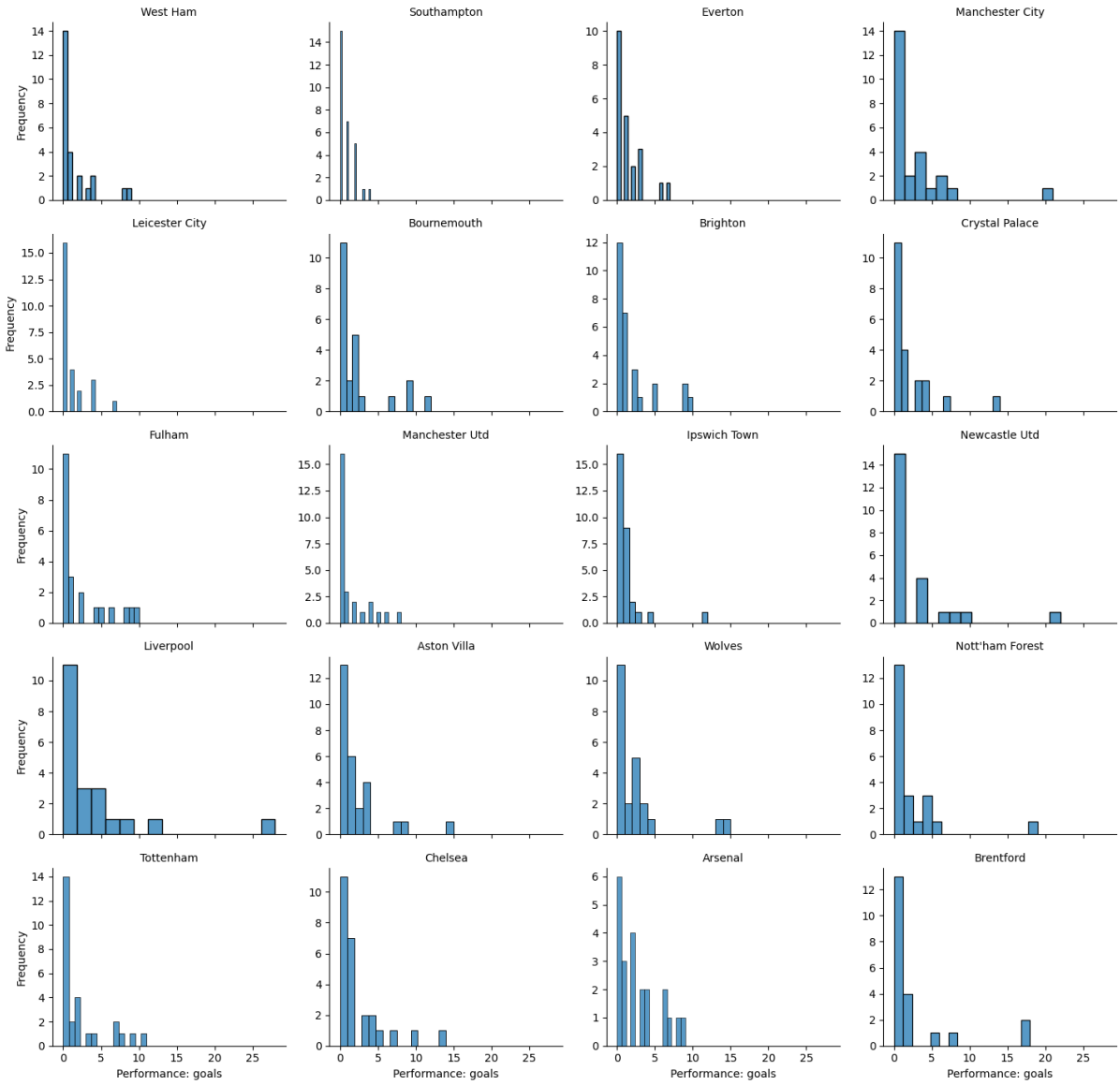
Kết quả và Nhận xét: Vì kết quả vẽ ra bao gồm rất nhiều hình ảnh, nên em sẽ chọn ra một chỉ số tiêu biểu để phân tích trong báo cáo. Chi tiết các hình ảnh kết quả trong thư mục plots trên Github.

Đánh giá kết quả của chỉ số Performance: goals (Bàn thắng) vì bàn thắng là yếu tố quyết định kết quả trận đấu và thứ hạng đội bóng, đồng thời số bàn thắng là chỉ số phổ biến nhất để đánh giá hiệu suất cầu thủ tấn công.



Hình 2.2: Biểu đồ phân bố Performance goals của cầu thủ

Phân phối số bàn thắng có xu hướng lệch phải rõ rệt, cho thấy phần lớn cầu thủ ghi rất ít bàn, trong khi chỉ một số ít ghi nhiều bàn.



Hình 2.3: Biểu đồ của Performance goals của các đội

Xu hướng tương tự được quan sát ở từng đội, với phần lớn cầu thủ ghi ít bàn. Một số đội như Manchester City và Arsenal có nhiều cầu thủ ghi bàn hơn mức trung bình, phản ánh khả năng tấn công mạnh mẽ hơn.

2.6 Xác định đội bóng dẫn đầu theo từng chỉ số và đội bóng xuất sắc nhất

Một trong những yêu cầu của bài tập là xác định đội bóng có điểm số cao nhất cho mỗi chỉ số thống kê và từ đó tìm ra đội bóng có hiệu suất tổng thể tốt nhất.

Trong quá trình xác định đội bóng xuất sắc nhất tổng thể, không phải tất cả các chỉ số đều được coi trọng như nhau. Một số chỉ số, nếu có giá trị cao, lại phản ánh hiệu suất không tốt hoặc những khía cạnh tiêu cực của một đội bóng. Để đảm bảo việc đánh giá

đội mạnh nhất dựa trên các yếu tố tích cực, cem đã định nghĩa một danh sách các "chỉ số tiêu cực"(`NEGATIVE_STATS`) trong file cấu hình `config_part2.py`.

Các chỉ số này được loại trừ khi tổng hợp số lần một đội đứng đầu để tìm ra đội có hiệu suất tổng thể tốt nhất.

Hàm `print_top_team_per_statistic` trong module `analysis.py` được sử dụng để thực hiện nhiệm vụ này.

```python
# Trích đoạn từ analysis.py - Hàm print_top_team_per_statistic
def print_top_team_per_statistic(summary_df, relevant_stats):
    # Lọc ra các đội, bỏ qua dòng tổng hợp 'all'
    teams_df = summary_df[summary_df['Team'] != 'all'].copy()
    # Danh sách để lưu trữ đội top cho mỗi chỉ số tích cực
    top_teams_for_positive_stats = []
    # Danh sách các chỉ số tích cực đã được xem xét
    positive_stats_considered = []
    # Xác định đội bóng tốt nhất cho từng chỉ số
    for stat in relevant_stats:
        col = f'Mean of {stat}' # Cột chứa giá trị trung bình của chỉ số

        # Chuyển đổi cột sang dạng số, loại bỏ NaN
        teams_df[col] = pd.to_numeric(teams_df[col], errors='coerce')
        valid_df = teams_df.dropna(subset=[col])

        # Tìm đội có giá trị trung bình cao nhất cho chỉ số hiện tại
        top_row = valid_df.loc[valid_df[col].idxmax()]
        top_team = top_row['Team']
        top_value = top_row[col]
        print(f"Top team for '{stat}': {top_team} (mean = {top_value:.3f})")

        # NEGATIVE_STATS được định nghĩa trong config_part2.py
        if stat not in NEGATIVE_STATS:
            top_teams_for_positive_stats.append(top_team)
            positive_stats_considered.append(stat)

    # Xác định đội bóng xuất sắc nhất (xuất hiện nhiều nhất trong top các chỉ số tích cực)
    top_team_series = pd.Series(top_teams_for_positive_stats)
    most_common_team = top_team_series.value_counts().idxmax()
    most_common_count = top_team_series.value_counts().max()
    # Hàm in ra màn hình kết quả
```

```
33      print(f"\nOverall best performing team (most frequent top team in
        ↪  non-negative statistics): {most_common_team} (appeared
        ↪  {most_common_count} times / {len(positive_stats_considered)}
        ↪  non-negative statistics considered)")
```

Hàm này nhận đầu vào là `summary_df` (DataFrame chứa thống kê tóm tắt đã tính ở bước trước) và `relevant_stats` (danh sách các chỉ số cần phân tích).

1. **Xác định đội bóng dẫn đầu theo từng chỉ số:**

   Đầu tiên, hàm lọc bỏ dòng tổng hợp "all"để chỉ xét dữ liệu của các đội. Với mỗi chỉ số trong `relevant_stats`:

   Hàm tìm cột giá trị trung bình tương ứng (ví dụ: 'Mean of Performance: goals'). Sau đó, nó xác định đội có giá trị trung bình cao nhất cho chỉ số đó và in ra kết quả. Điều này cho biết đội nào đang thể hiện tốt nhất ở khía cạnh cụ thể đó.

2. **Xác định đội bóng xuất sắc nhất tổng thể:** Trong quá trình duyệt qua các chỉ số, nếu một chỉ số không được coi là "tiêu cực"(ví dụ: số thẻ phạt, số lỗi – được định nghĩa trong biến `NEGATIVE_STATS` từ file `config_part2.py`), đội đứng đầu chỉ số đó sẽ được ghi nhận. Cuối cùng, hàm thống kê xem đội nào xuất hiện nhiều lần nhất trong danh sách các đội đứng đầu các chỉ số tích cực. Đội xuất hiện nhiều nhất được coi là Đội bóng xuất sắc nhất tổng thể và được in ra cùng với số lần xuất hiện trên tổng số các chỉ số tích cực đã được xem xét. Đây là cơ sở để phân tích đội bóng có hiệu suất toàn diện tốt nhất giải đấu.

Kết quả và nhận xét:

Từ kết quả dưới, có thể thấy Liverpool là đội dẫn đầu ở nhiều chỉ số quan trọng, đặc biệt là các chỉ số liên quan đến tấn công và kiểm soát bóng. Manchester City cũng nổi bật ở các chỉ số liên quan đến chuyền bóng và kiểm soát ở 1/3 sân đối phương. Crystal Palace và Brentford cho thấy sự hiệu quả ở một số khía cạnh phòng ngự.

Quan trọng nhất, khi xét đến các chỉ số không tiêu cực, Liverpool là đội xuất hiện nhiều lần nhất ở vị trí dẫn đầu, với 26 lần trong tổng số 63 chỉ số tích cực được xem xét. Điều này cho thấy Liverpool là đội bóng có hiệu suất tổng thể ấn tượng nhất trong giải đấu dựa trên phân tích này.

Dưới đây là kết quả được trích xuất từ việc chạy hàm `_top_team_per_statistic` :

Playing Time	
Matches played	Liverpool (24.476)
Starts	Brentford (17.810)
Minutes	Liverpool (1596.381)
Performance	
Goals	Liverpool (3.762)
Assists	Liverpool (2.810)
Yellow cards	Bournemouth (3.783)
Red cards	Arsenal (0.227)
Expected	
xG	Liverpool (3.629)
xAG	Liverpool (2.629)
Progression	
PrgC	Manchester City (40.560)
PrgP	Liverpool (81.381)
PrgR	Liverpool (80.619)
Per 90 minutes	
Gls	Manchester City (0.183)
Ast	Liverpool (0.148)
xG	Aston Villa (0.193)
xAG	Chelsea (0.153)
Goalkeeping: Performance	
GA90	Leicester City (2.730)
Save%	Bournemouth (80.000)
CS%	Brentford (59.100)
Goalkeeping: Penalty Kicks	
Save%	Everton (100.000)
Shooting: Standard	
SoT%	Nott'ham Forest (38.990)
SoT/90	Fulham (0.544)
G/Sh	Arsenal (0.137)
Dist	Nott'ham Forest (19.090)
Passing: Total	
Cmp	Liverpool (778.095)
Cmp%	Manchester City (86.548)
TotDist	Liverpool (13238.143)
Passing: By Distance	
Short Cmp%	Manchester City (92.152)
Medium Cmp%	Manchester City (89.580)
Long Cmp%	Liverpool (60.324)
Passing: Expected	
KP	Liverpool (22.000)
1/3	Liverpool (67.714)
PPA	Liverpool (18.619)
CrsPA	Fulham (4.227)
PrgP	Liverpool (81.381)
Goal and Shot Creation: SCA	
SCA	Liverpool (49.810)
SCA90	Liverpool (2.633)
Goal and Shot Creation: GCA	
GCA	Liverpool (6.476)
GCA90	Liverpool (0.348)
Defensive Actions: Tackles	
Tkl	Crystal Palace (32.619)
TklW	Crystal Palace (19.143)
Defensive Actions: Challenges	
Att	Liverpool (28.286)
Lost	Crystal Palace (14.048)
Defensive Actions: Blocks	
Blocks	Crystal Palace (20.476)
Sh	Brentford (8.381)
Pass	Crystal Palace (14.571)
Int	Bournemouth (14.000)
Possession: Touches	
Touches	Liverpool (1102.048)
Def Pen	Brentford (142.619)
Def 3rd	Brentford (357.333)
Mid 3rd	Liverpool (497.524)
Att 3rd	Manchester City (343.480)
Att Pen	Liverpool (55.810)
Possession: Take-Ons	
Att	Arsenal (29.727)
Succ%	Liverpool (54.910)
Tkld%	Leicester City (48.304)
Possession: Carries	
Carries	Manchester City (643.480)
PrgDist	Manchester City (1994.160)
PrgC	Manchester City (40.560)
1/3	Manchester City (30.360)
CPA	Manchester City (14.040)
Mis	Nott'ham Forest (23.000)
Dis	Newcastle Utd (17.652)
Possession: Receiving	
Rec	Liverpool (769.667)
PrgR	Newcastle Utd (17.652)
Miscellaneous: Performance	
Fls	Bournemouth (19.826)
Fld	Newcastle Utd (17.609)
Off	Nott'ham Forest (3.727)
Crs	Fulham (36.818)
Recov	Bournemouth (71.435)
Miscellaneous: Aerial Duels	
Won	Brentford (26.762)
Lost	Crystal Palace (27.143)
Won%	Southampton (54.172)

Top: Liverpool (appeared 26 times / 63 non-negative statistics considered)

2.7 Quy trình thực thi chính

Hàm `main` trong module `MAIN_part2.py` điều phối toàn bộ quy trình phân tích dữ liệu ở Chương 2. Quy trình này được thiết kế để thực hiện một chuỗi các bước một cách tuần tự, từ tải dữ liệu đến tạo ra các kết quả phân tích và trực quan hóa.

```
# Trích đoạn từ MAIN_part2.py - Hàm main
def main():
    # 1. Tải và tiền xử lý dữ liệu
    df, numeric_cols = load_data()

    # 2. Tìm Top/Bottom 3 cầu thủ cho mỗi chỉ số
    find_top_bottom_players_all_stats(df, numeric_cols)

    # 3. Tính toán và lưu bảng tóm tắt thống kê theo đội
    summary_df = generate_statistics_summary(df, numeric_cols)

    # 4. Xác định đội dẫn đầu từng chỉ số và đội xuất sắc nhất
    # (Sử dụng summary_df từ bước 3)
    print_top_team_per_statistic(summary_df, numeric_cols)

    # 5. Tạo và lưu các biểu đồ histogram
    generate_histograms(df)
```

Mô tả các bước thực hiện:

Hàm `main` tuần tự thực hiện các chức năng sau:

1. **Tải dữ liệu:** Đọc dữ liệu từ file `results.csv` (từ Chương 1), chuyển đổi các cột số liệu cần thiết sang dạng số.

2. **Tìm Top/Bottom 3 cầu thủ :** Sử dụng hàm `get_top_bottom_players` từ `analysis.py` để xác định 3 cầu thủ hàng đầu và 3 cầu thủ cuối bảng cho mỗi chỉ số. Kết quả được lưu vào file `top_3_txt`.

3. **Tạo bảng tóm tắt thống kê:** Gọi hàm `calculate_stats_summary` từ `analysis.py` để tính toán các thông số thống kê (trung bình, trung vị, độ lệch chuẩn) theo từng đội. Bảng tóm tắt này được lưu vào file `results2.csv`.

4. **Xác định đội dẫn đầu và đội xuất sắc nhất:** Sử dụng `summary_df` (từ bước 3) và hàm `print_top_team_per_statistic` từ `analysis.py` để tìm ra đội bóng dẫn đầu cho từng chỉ số và đội có hiệu suất tổng thể tốt nhất dựa trên các chỉ số tích cực. Kết quả được in ra console.

5. **Tạo biểu đồ Histograms:** Sử dụng các hàm vẽ từ `plotting.py` để vẽ biểu đồ phân phối cho các chỉ số trong `SELECTED_STATS`. Các biểu đồ được lưu trong thư mục `plots`.

Chương 3

Phân cụm Cầu thủ bằng K-Means và PCA

Trong phần này, em sẽ trình bày quá trình áp dụng thuật toán phân cụm K-Means để nhóm các cầu thủ bóng đá Ngoại hạng Anh mùa giải 2024-2025 thành các nhóm có đặc điểm thống kê tương đồng. Tiếp đó, em sử dụng kỹ thuật Phân tích Thành phần Chính (PCA) để giảm chiều dữ liệu, giúp trực quan hóa các cụm cầu thủ trên biểu đồ 2D, qua đó làm rõ hơn mối quan hệ và sự khác biệt giữa các nhóm.

3.1 Giới thiệu về K-Means và lựa chọn số cụm (K)

K-Means là một thuật toán học không giám sát phổ biến, dùng để phân nhóm dữ liệu. Thuật toán hoạt động bằng cách chia N điểm dữ liệu vào K cụm sao cho mỗi điểm gần tâm cụm của nó nhất. Mục tiêu chính là giảm tổng bình phương khoảng cách từ các điểm đến tâm cụm – chỉ số này còn gọi là Inertia[3].

Để xác định số cụm tối ưu (K) cho bộ dữ liệu cầu thủ, em đã sử dụng phương pháp "Elbow" (Khuỷu tay). Phương pháp này chạy K-Means với nhiều giá trị K (từ 2 đến 15), tính Inertia cho từng K và vẽ biểu đồ. Từ biểu đồ, điểm "khuỷu tay" – nơi mà Inertia không còn giảm đáng kể dù K tăng – sẽ được chọn làm số cụm tối ưu.

Lí do cho lựa chọn giá trị K được chạy từ 2 đến 15 là vì một quy tắc kinh nghiệm đề xuất rằng số cụm tối đa, k, nên được chọn xấp xỉ theo công thức $k \approx \sqrt{n/2}$, với n là tổng số điểm dữ liệu [6]. Cụ thể trong phần này $N = 491$, với 491 là số lượng cầu thủ ghi nhận được ở kết quả của Chương 1. Theo quy tắc trên, giá trị k tối đa nên thử nghiệm là $k \approx \sqrt{491/2} \approx 15.67$, do đó việc giới hạn K trong đoạn từ 2 đến 15 là hợp lý.

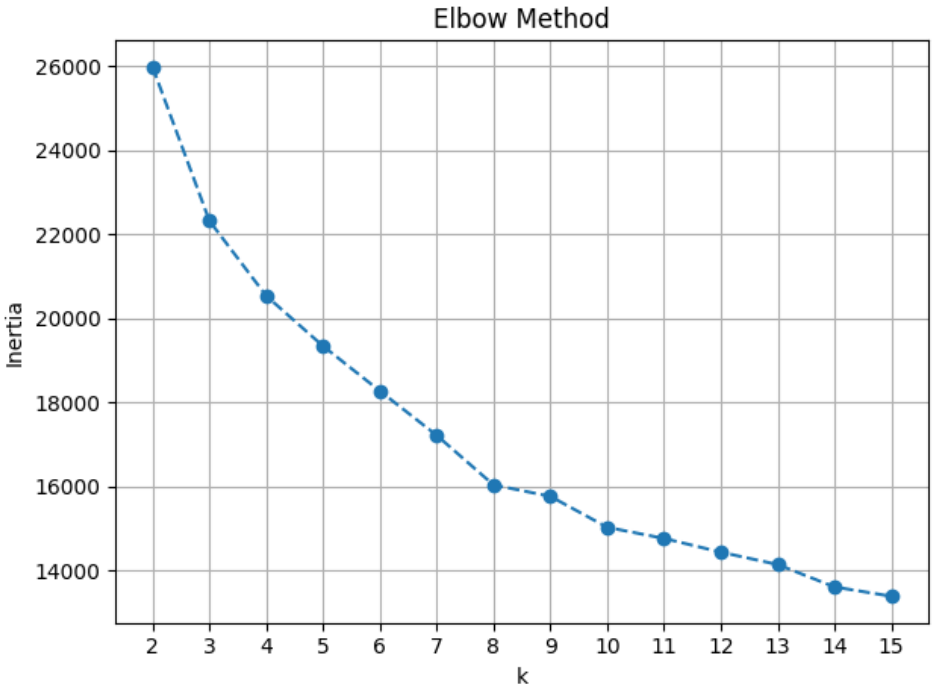
```
# Trích đoạn từ MAIN_part3.py - Hàm plot_elbow_method

def plot_elbow_method(df_scaled, k_range, output_path):
    # Tính inertia (tổng khoảng cách các điểm đến tâm cụm) cho mỗi giá trị k
    inertia = [KMeans(n_clusters=k, random_state=42,
    ↪  n_init='auto').fit(df_scaled).inertia_ for k in k_range]

    # Vẽ biểu đồ Elbow để chọn số cụm tối ưu
    plt.plot(k_range, inertia, 'o--')
    plt.xlabel('k'); plt.ylabel('Inertia'); plt.title('Elbow Method')
    plt.xticks(k_range); plt.grid(True); plt.tight_layout()
    plt.savefig(output_path); plt.close()

```

Kết quả:Dưới đây là biểu đồ Elbow thu được từ quá trình phân tích:



Hình 3.1: Biểu đồ Elbow Method

Quan sát hình trên, em nhận thấy từ K=2 đến K=4, giá trị Inertia giảm rất nhanh. Sự suy giảm này vẫn đáng kể khi K tăng từ 4 đến 6. Tuy nhiên, sau K=6, đường cong bắt đầu thoải dần, và việc tăng thêm số cụm không còn mang lại sự giảm mạnh mẽ cho Inertia nữa. Mặc dù có thể có một "khuỷu tay"khác nhẹ nhàng hơn ở K=8, nhưng để cân bằng giữa việc có đủ số cụm để phân biệt các nhóm cầu thủ và tránh làm mô hình trở nên quá phức tạp, em quyết định chọn K=6 là số cụm tối ưu cho phân tích này.

Để thực hiện phân cụm đã sử dụng toàn bộ các chỉ số thống kê dạng số trong bộ dữ liệu, sau khi loại bỏ các thông tin mang tính định danh như tên cầu thủ, quốc tịch, đội bóng, vị trí thi đấu và độ tuổi. Các chỉ số này phản ánh nhiều khía cạnh khác nhau về hiệu suất thi đấu của cầu thủ. Trước khi đưa vào mô hình K-Means, dữ liệu đã được chuẩn hóa bằng phương pháp `StandardScaler` nhằm đảm bảo sự cân bằng giữa các thang đo, tránh tình trạng một vài chỉ số chi phối kết quả phân cụm. Việc co giãn đặc trưng là cần thiết cho các thuật toán dựa trên khoảng cách như K-Means để tránh các đặc trưng có thang đo lớn hơn chi phối kết quả phân cụm [7]. Ngoài ra, tập biến `KEY_STATS_FOR_INTERPRETATION` (gồm 18 chỉ số) sẽ được dùng để mô tả và giải thích các cụm sau khi phân tích.

```
# Trích đoạn từ MAIN_part3.py - Hàm run_kmeans và lựa chọn OPTIMAL_K
OPTIMAL_K = 6 # Đã chọn

def run_kmeans(df_scaled, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init='auto')
    cluster_labels = kmeans.fit_predict(df_scaled)
    return cluster_labels
```

3.2 Kết quả phân cụm K-Means

Sau khi chạy thuật toán K-Means với K=6, các cầu thủ trong giải đấu đã được phân vào 6 nhóm khác nhau. Dưới đây là tóm tắt phân phối số lượng cầu thủ trong mỗi cụm và các đặc điểm chính của từng cụm.

Phân phối số lượng cầu thủ theo cụm:

Cluster 0: 181 cầu thủ (36.9%)

Cluster 1: 97 cầu thủ (19.8%)

Cluster 2: 50 cầu thủ (10.2%)

Cluster 3: 95 cầu thủ (19.3%)

Cluster 4: 41 cầu thủ (8.4%)

Cluster 5: 27 cầu thủ (5.5%)

Đặc điểm chi tiết của từng cụm:

Cụm 0: Gồm phần lớn là hậu vệ (DF). Nhìn chung, nhóm này có vẻ thiên về phòng ngự truyền thống, ít tham gia vào tấn công và xây dựng lối chơi.

Cụm 1: Chủ yếu là tiền vệ (MF). Đây có vẻ là những tiền vệ năng nổ, giỏi thu hồi bóng, tranh chấp và đóng góp vào việc phát triển bóng lên phía trên, nhưng không mạnh ở khâu ghi bàn.

Cụm 2: Bao gồm tiền đạo (FW) và tiền vệ (MF). Nhóm này nổi bật ở nhiều mặt, từ phòng ngự, phát triển bóng cho đến tấn công và ghi bàn. Đây có thể là những cầu thủ tấn công toàn diện.

Cụm 3: Gồm tiền đạo (FW) và tiền vệ (MF). Nhóm này có vẻ yếu ở khả năng phòng ngự và phát triển bóng, nhưng lại có chỉ số tấn công (kiến tạo, bàn thắng, sút trúng đích) ở mức cao. Tuy nhiên, xG (bàn thắng kỳ vọng) thấp cho thấy họ có thể dứt điểm tốt hoặc tận dụng cơ hội từ những tình huống không quá tốt.

Cụm 4: Hầu hết là hậu vệ (DF). Đây có vẻ là những hậu vệ giỏi phòng ngự, tranh chấp và có khả năng chuyền bóng phát triển tấn công, nhưng ít tự mình mang bóng lên phía trên hay tham gia trực tiếp vào các pha bóng cuối cùng.

Cụm 5: Chủ yếu là tiền đạo (FW). Nhóm này có vẻ ít tham gia phòng ngự và chuyền bóng phát triển lối chơi, nhưng rất mạnh mẽ trong vòng cấm đối phương, giỏi qua người, kiến tạo, ghi bàn và có các chỉ số tấn công tốt. Đây có thể là những tiền đạo cắm điển hình.

Nhận xét chung về kết quả phân cụm:

Việc phân thành 6 cụm đã cho thấy sự khác biệt tương đối rõ ràng giữa các nhóm cầu thủ dựa trên vai trò và phong cách chơi của họ.

Cụm 0 chiếm số lượng lớn nhất, điều này cũng dễ hiểu vì trong một đội hình luôn cần nhiều cầu thủ đảm nhận vai trò phòng ngự. Ngược lại, các cụm chuyên biệt hơn như Cụm 2 hay Cụm 5 có số lượng ít hơn, cho thấy đây là những vai trò đòi hỏi kỹ năng đặc thù và không phải cầu thủ nào cũng đáp ứng được.

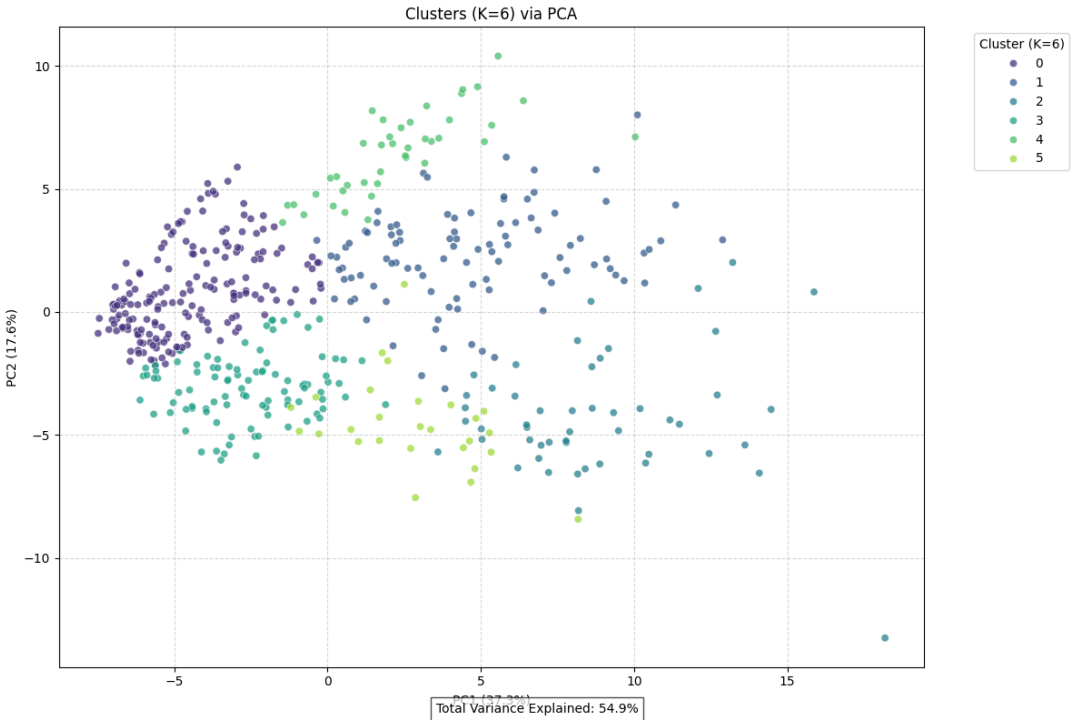
3.3 Trực quan hóa các cụm bằng PCA

Để trực quan hóa các cụm cầu thủ K-Means, em đã áp dụng Phân tích Thành phần Chính (PCA). PCA giúp giảm chiều dữ liệu đặc trưng (đã chuẩn hóa) của cầu thủ xuống còn 2 thành phần chính (PC1 và PC2). Kết quả là một biểu đồ phân tán 2D, với mỗi điểm là một cầu thủ và màu sắc thể hiện cụm của cầu thủ đó.

```python
# Trích đoạn từ MAIN_part3.py - Hàm visualize_pca
def visualize_pca(df_scaled_with_clusters, df_identifiers, k, output_path):
    # Giảm chiều dữ liệu bằng PCA và trực quan hóa các cụm
    pca = PCA(n_components=2)
    comps = pca.fit_transform(data.drop(columns='Cluster'))
    data_pca = pd.DataFrame(comps, columns=['PC1', 'PC2'])
    data_pca['Cluster'] = data['Cluster']

    # Vẽ scatter plot các cụm theo hai thành phần chính
    sns.scatterplot(data=data_pca, x='PC1', y='PC2', hue='Cluster')
    plt.title(f'PCA Clustering (K={k})')
    plt.savefig(output_path)
```

Kết quả biểu đồ phân tán 2D:



Hình 3.2: Biểu đồ phân tán 2D các cụm cầu thủ sau khi áp dụng PCA (K=6)

Nhận xét về biểu đồ PCA: Biểu đồ PCA (Hình trên) cho thấy sự phân bố của 6 cụm cầu thủ trong không gian 2 chiều được tạo bởi hai thành phần chính đầu tiên.

Hai thành phần này giải thích được tổng cộng 54.9% phương sai của dữ liệu gốc (PC1 giải thích 37.3%, PC2 giải thích 17.6%). Từ biểu đồ, em có thể quan sát:

- Có sự tách biệt nhất định giữa các cụm, mặc dù một số cụm có vùng chồng lấn.

  Ví dụ, Cụm 0 (màu tím đậm) chiếm một vùng không gian rộng và có phần tách biệt, phản ánh nhóm đa số là hậu vệ với các đặc điểm khá riêng biệt.

- Các cụm cầu thủ tấn công có xu hướng nằm ở các vùng khác nhau trên biểu đồ, cho thấy sự đa dạng trong vai trò tấn công.

  Ví dụ, Cụm 2 (màu xanh lá cây nhạt) và Cụm 5 (màu vàng) có thể chiếm các vị trí khác nhau, phản ánh sự khác biệt trong bộ kỹ năng của họ dù đều là cầu thủ tấn công.

Nhìn chung, biểu đồ PCA giúp cung cấp một cái nhìn trực quan, hỗ trợ cho việc hiểu rõ hơn về cấu trúc của dữ liệu và mối quan hệ tương đối giữa các nhóm cầu thủ đã được K-Means phân loại. Dù chỉ giữ lại được một phần phương sai của dữ liệu gốc, việc giảm chiều xuống 2D vẫn hữu ích trong việc phân tích và trình bày kết quả.

Chương 4

Ước tính giá trị cầu thủ

Phần này tập trung vào việc thu thập dữ liệu giá trị chuyển nhượng của các cầu thủ Ngoại hạng Anh mùa giải 2024-2025 và đề xuất một phương pháp để ước tính giá trị của họ dựa trên các số liệu thống kê đã thu thập. Mục tiêu là xây dựng một mô hình có khả năng dự đoán giá trị thị trường của cầu thủ, một thông tin quan trọng trong việc đánh giá và quản lý đội bóng.

4.1 Cấu trúc chương trình

Để thực hiện các yêu cầu của Chương 4, chương trình được tổ chức thành các module sau:

`MAIN_part4.py`: Module chính điều phối quá trình thu thập dữ liệu giá trị chuyển nhượng và xử lý dữ liệu.

`scraper_part4.py` : Chứa các hàm để truy cập và trích xuất thông tin giá trị chuyển nhượng của cầu thủ từ trang web footballtransfers.com.

`processor_part4.py` : Chịu trách nhiệm xử lý dữ liệu thô đã thu thập, đặc biệt là lọc các cầu thủ dựa trên tiêu chí thời gian thi đấu (>900 phút) từ kết quả của Chương 1.

`config_part4.py` : Lưu trữ các cấu hình cần thiết cho việc scraping như URL, các CSS selectors để trích xuất dữ liệu, ngưỡng thời gian thi đấu tối thiểu, và tên các tệp đầu ra.

`training_pipeline.py` : Module này chứa toàn bộ quy trình xây dựng mô hình ước tính giá trị cầu thủ, bao gồm tải dữ liệu, tiền xử lý, lựa chọn đặc trưng, huấn luyện mô hình và đánh giá.

4.2 Thu thập dữ liệu giá trị chuyển nhượng

4.2.1 Lựa chọn công cụ và nguồn dữ liệu

Dữ liệu giá trị chuyển nhượng ước tính (Estimated Transfer Value - ETV) của cầu thủ được thu thập từ trang footballtransfers.com. Tương tự cho Chương 1, em sử dụng `Selenium`, và `BeautifulSoup` sau đó được dùng để phân tích cú pháp HTML của trang.

4.2.2 Chi tiết thực hiện

Quá trình thu thập dữ liệu được thực hiện chủ yếu bởi hàm `scrape_transfer_data` trong `scraper_part4.py`. Logic chính của hàm này bao gồm:

1. **Truy cập URL chính** : Điều hướng đến trang danh sách cầu thủ của giải Ngoại hạng Anh trên footballtransfers.com(được định nghĩa trong `TRANSFER_URL` ở `config_part4.py`).

2. **Lặp qua các trang** : Sử dụng Selenium để tự động nhấn nút chuyển trang (sử dụng `NEXT_PAGE_SELECTOR`) để tải danh sách cầu thủ từ tất cả các trang.

3. **Trích xuất thông tin cơ bản** : Từ mỗi dòng dữ liệu của cầu thủ (xác định bởi `PLAYER_ROW_SELECTOR`), trích xuất các thông tin:

   - Tên cầu thủ (`PLAYER_NAME_SELECTOR`)
   - Đội bóng (`TEAM_NAME_SELECTOR`)
   - Tuổi (`AGE_SELECTOR`)
   - Vị trí (`POSITION_SELECTOR`)
   - Giá trị chuyển nhượng ước tính (ETV) hiện tại (`ETV_SELECTOR`)
   - URL trang cá nhân của cầu thủ.

4. **Thu thập giá trị ETV cao nhất** : Đối với mỗi cầu thủ, truy cập URL trang cá nhân của họ. Từ trang cá nhân, trích xuất giá trị ETV cao nhất từng được ghi nhận (sử dụng `HIGHEST_ETV_SELECTOR_PROFILE`). Việc này được thực hiện bởi hàm `scrape_highest_etv`.

5. **Làm sạch dữ liệu** : Giá trị ETV (ví dụ: "€10.5m", "€500k") được làm sạch và chuyển đổi sang dạng số (triệu EUR) bằng hàm `clean_value`.

6. **Lưu dữ liệu thô** : Dữ liệu thu thập được lưu vào tệp `raw_data_with_highest_etv.csv` (định nghĩa trong `RAW_DATA_FILENAME`).

Đoạn code minh họa logic chính trong `scraper_part4.py` (hàm `scrape_transfer_data` và `scrape_highest_etv`):

```
def scrape_transfer_data(driver: WebDriver) -> bool:
    # Khởi tạo các biến lưu trữ dữ liệu và theo dõi tiến trình
    data, scraped_keys = [], set()
    page = 1
    # Mở trang web danh sách chuyển nhượng
    driver.get(TRANSFER_URL)
    WebDriverWait(driver, 20).until(EC.presence_of_element_located((By.CSS_SELECTOR,
    ↪  PLAYER_TABLE_SELECTOR)))
    # Lặp qua các trang để lấy dữ liệu
    while True:
        # Đợi bảng cầu thủ xuất hiện
        WebDriverWait(driver, WAIT_TIME).until(
            EC.visibility_of_element_located((By.CSS_SELECTOR, PLAYER_ROW_SELECTOR))
        )
        # Phân tích dữ liệu trang
        soup = BeautifulSoup(driver.page_source, 'html.parser')
        rows = soup.select(PLAYER_ROW_SELECTOR)
        # Bỏ qua nếu không tìm thấy dữ liệu
        if not rows:
            driver.refresh()
            continue
        # Xử lý từng hàng dữ liệu cầu thủ
        added = 0
        for row in rows:
            name = safe_get(row, PLAYER_NAME_SELECTOR)
```

```
            team = safe_get(row, TEAM_NAME_SELECTOR)
            url = safe_get_attribute(row, PLAYER_NAME_SELECTOR, 'href')
            if not (name and team and url): continue
            # Kiểm tra trùng lặp
            key = (name.strip(), team.strip())
            if key in scraped_keys: continue
            # Tạo thông tin cầu thủ
            player = {
                'Player': key[0],
                'Team_TransferSite': key[1],
                'Age': safe_get(row, AGE_SELECTOR),
                'Position': safe_get(row, POSITION_SELECTOR),
                TARGET_VARIABLE: clean_value(safe_get(row, ETV_SELECTOR)),
                'Profile_URL': url,
                'Highest_ETV': None
            }
            # Thêm vào danh sách
            data.append(player)
            scraped_keys.add(key)
            added += 1
        # Chuyển đến trang tiếp theo nếu có
        try:
            next_btn = WebDriverWait(driver, WAIT_TIME).until(
                EC.element_to_be_clickable((By.CSS_SELECTOR, NEXT_PAGE_SELECTOR))
            )
            driver.execute_script("arguments[0].click();", next_btn)
            time.sleep(random.uniform(2, 4))
            page += 1
        except:
            # Kết thúc khi không còn trang nào
            break
    # Thu thập thông tin Highest_ETV từ trang cá nhân
    for i, player in enumerate(data):
        url = player.get('Profile_URL')
        player['Highest_ETV'] = scrape_highest_etv(driver, url) if url else None
        time.sleep(random.uniform(0.5, 1.5))
    # Xóa dữ liệu URL không cần thiết
    for p in data:
        p.pop('Profile_URL', None)
    # Lưu dữ liệu đã thu thập
    return save_data(data)
```

```
def scrape_highest_etv(driver: WebDriver, url: str) -> float | None:
    if not url: return None
    # Mở trang profile
    driver.get(url)
    WebDriverWait(driver,
    ↪  WAIT_TIME).until(EC.presence_of_element_located((By.CSS_SELECTOR,
    ↪  PROFILE_PAGE_LOAD_CHECK_SELECTOR)))
```

```
    time.sleep(random.uniform(1, 2))
    # Lấy giá trị ETV cao nhất
    etv_text = safe_get(BeautifulSoup(driver.page_source, 'html.parser'),
    ↪  HIGHEST_ETV_SELECTOR_PROFILE)
    return clean_value(etv_text) if etv_text else None
```

4.3 Xử lý dữ liệu

4.3.1 Lọc cầu thủ

Sau khi thu thập dữ liệu giá trị chuyển nhượng thô, bước tiếp theo là xử lý và lọc dữ liệu này. Hàm `process_transfer_data` trong `processor_part4.py` thực hiện nhiệm vụ này.

Lấy danh sách cầu thủ hợp lệ : Hàm `get_valid_players_from_part1` được gọi để đọc tệp `results.csv` từ Chương 1. Từ đó, trích xuất danh sách các cầu thủ có thời gian thi đấu (Playing Time: minutes) lớn hơn ngưỡng `MIN_MINUTES_THRESHOLD` (900 phút) được định nghĩa trong `config_part4.py`.

Lọc dữ liệu chuyển nhượng : DataFrame chứa dữ liệu chuyển nhượng thô được lọc để chỉ giữ lại những cầu thủ có tên nằm trong danh sách cầu thủ hợp lệ đã lấy ở bước trên.

Lưu kết quả : Dữ liệu đã lọc được lưu vào tệp `estimation_data_with_highest_etv.csv`

Đoạn code minh họa logic chính trong `processor_part4.py` (hàm `get_valid_players_from_part1` và `process_transfer_data`):

```
def get_valid_players_from_part1() -> set:
    # Đọc dữ liệu từ file kết quả phần 1
    df = pd.read_csv(PART1_RESULTS_FILE)
    # Chuyển đổi cột phút thành số và xóa dấu phẩy
    df['minutes'] =
    ↪  pd.to_numeric(df[PART1_MINUTES_COLUMN].astype(str).str.replace(',', ''),
    ↪  errors='coerce')
    # Lọc cầu thủ có số phút chơi lớn hơn ngưỡng tối thiểu
    players = set(df.loc[df['minutes'] > MIN_MINUTES_THRESHOLD, PART1_PLAYER_COLUMN]
                  .dropna().astype(str).str.strip())
    return players
```

```
def process_transfer_data() -> pd.DataFrame | None:
    # Lấy danh sách cầu thủ hợp lệ từ phần 1
    valid_players = get_valid_players_from_part1()
    # Đọc dữ liệu thô từ file
    df = pd.read_csv(RAW_DATA_FILENAME)
    # Chuẩn hóa tên cầu thủ
    df['Player'] = df['Player'].astype(str).str.strip()
    # Lọc dữ liệu chỉ giữ lại các cầu thủ hợp lệ
    filtered = df[df['Player'].isin(valid_players)].copy()
    # Tạo thư mục đầu ra nếu chưa tồn tại
    os.makedirs(OUTPUT_FOLDER, exist_ok=True)
```

```
    # Lưu dữ liệu đã lọc
    filtered.to_csv(ESTIMATION_READY_DATA_FILENAME, index=False, encoding='utf-8-sig')
    return filtered
```

4.3.2 Kết quả thu thập và xử lý

Dữ liệu cuối cùng sẵn sàng cho việc xây dựng mô hình ước tính giá trị được lưu trong tệp `estimation_data_with_highest_etv.csv`.

Tệp này chứa thông tin về tên cầu thủ, tuổi, vị trí, giá trị chuyển nhượng hiện tại (ETV) và giá trị ETV cao nhất từng có, chỉ bao gồm các cầu thủ đã thi đấu trên 900 phút trong mùa giải.

	A	B	C	D	E	F
1	Player	Team_TransferSite	Age	Position	TransferValue_EUR_Millions	Highest_ETV
2	Erling Haaland	Man City	24	F (C)	198.8	199.6
3	Martin Ødegaard	Arsenal	26	M, AM (C)	126.5	134.5
4	Alexander Isak	Newcastle Utd.	25	F (C)	120.3	120.3
5	Cole Palmer	Chelsea	22	M (C)	115.4	119
6	Declan Rice	Arsenal	26	M (C)	107.8	120
7	Alexis Mac Allister	Liverpool	26	M, DM, AM (C)	106.1	117
8	Phil Foden	Man City	24	AM (R), M (C)	105.7	138.6
9	Bukayo Saka	Arsenal	23	F, M (R)	101.3	127.5
10	Ryan Gravenberch	Liverpool	22	DM, M (C)	85.3	85.5
11	Bruno Guimarães	Newcastle Utd.	27	DM, M (C)	83.2	83.2
12	Moisés Caicedo	Chelsea	23	DM, M (C)	80.7	86.1
13	William Saliba	Arsenal	24	D (C)	79.5	79.5
14	Omar Marmoush	Man City	26	F (C)	79.1	79.1
15	Joško Gvardiol	Man City	23	D (CL)	78.7	86.6
16	Gabriel Magalhães	Arsenal	27	D (C)	75.5	75.5
17	Sávio	Man City	21	F (R), M (RL)	74.8	74.8
18	Enzo Fernández	Chelsea	24	M, DM (C)	73.7	82.7
19	Dominik Szoboszlai	Liverpool	24	M (C), AM (L)	71.4	71.4
20	Brennan Johnson	Tottenham	23	F, M (R)	71.3	71.3
21	Lucas Paquetá	West Ham United	27	AM, M (C)	71.2	75
22	Leny Yoro	Man Utd	19	D (CR)	71	71
23	Luis Díaz	Liverpool	28	F (CL), M (L)	71	78.5
24	Kai Havertz	Arsenal	25	F (C)	70.5	98.6
25	Morgan Rogers	Aston Villa	23	M (CR)	69	79.6
26	Murillo	Nottingham	22	D (C)	68.4	68.4
27	Cody Gakpo	Liverpool	25	F, M, AM (L)	67.5	72.2
28	Nicolas Jackson	Chelsea	23	F (C)	66.2	71.8
29	Kobbie Mainoo	Man Utd	20	M, DM (C)	66	66
30	Rico Lewis	Man City	20	D (R)	62.7	63.8
31	Matheus Cunha	Wolverhampton	25	F (C)	62.6	62.6
32	Gabriel Martinelli	Arsenal	23	F, AM (L)	62.6	70.1
33	Eberechi Eze	C. Palace	26	AM, M (C), F (L)	62.2	62.2
34	Amadou Onana	Aston Villa	23	DM, M (C)	62.1	64.5

estimation_data_with_highest_et

Hình 4.1: Dữ liệu ước tính giá trị chuyển nhượng trong csv

4.4 Đề xuất phương pháp ước tính giá trị cầu thủ

4.4.1 Giới thiệu

Mục tiêu là xây dựng một mô hình học máy có khả năng ước tính giá trị chuyển nhượng (ETV) của cầu thủ dựa trên các số liệu thống kê hiệu suất thi đấu (từ Chương 1) và các thông tin cơ bản khác (tuổi, vị trí). Biến mục tiêu là `TransferValue_EUR_Millions` (đã được làm sạch từ dữ liệu thu thập).

4.4.2 Lựa chọn đặc trưng

Việc lựa chọn đặc trưng là một bước quan trọng để xây dựng mô hình hiệu quả.

Kết hợp dữ liệu : Dữ liệu thống kê từ `results.csv` (Chương 1) được kết hợp với dữ liệu giá trị chuyển nhượng từ `estimation_data_with_highest_etv.csv` (Chương 4) dựa trên tên cầu thủ (Player).

Loại bỏ các cột định danh : Các cột định danh như `Player`, `Nation`, `Team`, (được liệt kê trong `ID_COLS` của `training_pipeline.py`) được loại bỏ khỏi tập đặc trưng đầu vào (X) vì chúng không mang tính tổng quát cho việc dự đoán hoặc có thể gây rò rỉ dữ liệu.

Xử lý cột số (Numerical Features) :

- Tất cả các cột chứa số liệu thống kê, tuổi, và `Highest_ETV` được coi là đặc trưng số.
- Các giá trị N/a hoặc các chuỗi không phải số được chuyển đổi thành NaN.
- Những cột có tỷ lệ giá trị thiếu lớn (cụ thể $> 50\%$) sẽ bị loại bỏ.

Xử lý cột hạng mục (Categorical Features) :

- Cột `Position` (và các cột khác nếu có sau khi đã chuyển đổi sang số) được coi là đặc trưng hạng mục (được quản lý bởi `CAT_COLS_BASE` và tự động phát hiện trong `training_pipeline.py`).
- Giá trị thiếu trong các cột hạng mục được điền bằng một giá trị cố định như 'Missing'.

Biến mục tiêu (Target Variable) : Giá trị `TransferValue_EUR_Millions` được sử dụng làm biến mục tiêu (y).

Để giảm thiểu ảnh hưởng của sự phân phối lệch và các giá trị ngoại lệ lớn, phép biến đổi logarit (`np.log1p`) được áp dụng cho biến mục tiêu.

Mô hình sẽ dự đoán giá trị log của ETV, sau đó giá trị dự đoán sẽ được chuyển đổi ngược (`np.expm1`) để có được ETV thực tế.

4.4.3 Lựa chọn mô hình

Dựa trên tính chất của bài toán là hồi quy – dự đoán một giá trị liên tục (giá trị cầu thủ) và sự phức tạp tiềm ẩn của mối quan hệ giữa các đặc trưng và giá trị cầu thủ, em đã tiến hành phân tích tổng hợp cùng với việc tham khảo các nghiên cứu trước đây. Cụ thể, nghiên cứu *"Football Player Value Prediction: Comparing Machine Learning Models with Cross-Validation"* [5] của tác giả Yitong Kong (Đại học Sorbonne, Paris, Pháp) sử dụng dữ liệu từ trang Kaggle và áp dụng phương pháp kiểm định chéo để đánh giá hiệu quả của các mô hình dự đoán. Kết quả cho thấy mô hình Gradient Boosting đạt hiệu suất tốt nhất, với chỉ số RMSE (Root Mean Square Error – Sai số bình phương trung bình gốc) thấp nhất là 0.450 và hệ số R-squared (hệ số xác định) cao nhất là 0.928.

Ngoài ra, nghiên cứu *"Comparing Machine Learning and Ensemble Learning in the Field of Football"*[4] cũng nhấn mạnh rằng các mô hình học máy tổ hợp, đặc biệt là Gradient Boosting và Random Forest, thường vượt trội hơn so với các mô hình học máy cơ bản nhờ khả năng giảm thiểu sai số dự đoán và cải thiện độ chính xác tổng thể. Nghiên cứu này chỉ ra rằng trong các bài toán có dữ liệu phức tạp, nhiều chiều, các mô hình học máy tổ hợp có ưu thế rõ rệt nhờ khả năng tổng hợp sức mạnh từ nhiều mô hình con, giảm thiểu hiện tượng overfitting và cải thiện khả năng khái quát hóa.

Từ những phân tích này, em quyết định lựa chọn mô hình **Gradient Boosting Regressor** (từ thư viện `sklearn.ensemble`) để thực hiện xử lý dữ liệu đề yêu cầu, đồng thời tiến hành và đánh giá hiệu suất thực tế của mô hình. Đây là một mô hình học máy mạnh mẽ, dựa trên nguyên lý của cây quyết định, có khả năng xử lý tốt các mối quan hệ phi tuyến tính và tương tác phức tạp giữa các đặc trưng. Mô hình này xây dựng các cây quyết định một cách tuần tự, mỗi cây mới sẽ cố gắng sửa lỗi của cây trước đó, từ đó giúp cải thiện độ chính xác của dự đoán. Nhờ vào tính năng học tuần tự này, **Gradient Boosting** có khả năng xử lý hiệu quả các bộ dữ liệu phức tạp và thường vượt trội hơn các mô hình hồi quy tuyến tính thông thường.

Quy trình xây dựng mô hình trong `training_pipeline.py` bao gồm:

Phân chia dữ liệu : Dữ liệu được chia thành tập huấn luyện (train) và tập kiểm tra (test) với tỷ lệ 80:20.

Tiền xử lý (Preprocessing Pipeline) : Một `ColumnTransformer` được sử dụng để áp dụng các bước tiền xử lý khác nhau cho các loại cột khác nhau:

- Đối với đặc trưng số (`num_cols`) :
  - `KNNImputer`: Điền giá trị thiếu bằng cách sử dụng thuật toán K-Nearest Neighbors.
  - `StandardScaler`: Chuẩn hóa các đặc trưng bằng cách loại bỏ giá trị trung bình và chia cho độ lệch chuẩn.
- Đối với đặc trưng hạng mục (`cat_cols`) :
  - `SimpleImputer`: Điền giá trị thiếu bằng giá trị xuất hiện nhiều nhất ('most_frequent').
  - `OneHotEncoder`: Chuyển đổi các đặc trưng hạng mục thành dạng số bằng phương pháp one-hot encoding, `handle_unknown='ignore'` để xử lý các giá trị mới có thể xuất hiện trong tập test.

Tối ưu hóa siêu tham số) : `RandomizedSearchCV` được sử dụng để tìm kiếm bộ siêu tham số tốt nhất cho `GradientBoostingRegressor`.

Các siêu tham số được tìm kiếm bao gồm `n_estimators`, `learning_rate`, `max_depth`, `min_samples_split`, `min_samples_leaf`, và `subsample`.

Việc tìm kiếm được thực hiện với 5-fold cross-validation và mục tiêu là tối thiểu hóa `neg_root_mean_squared_error`.

Huấn luyện mô hình : Mô hình `GradientBoostingRegressor` được huấn luyện trên tập huấn luyện đã qua tiền xử lý với bộ siêu tham số tốt nhất tìm được.

Đánh giá mô hình : Mô hình được đánh giá trên tập kiểm tra bằng các chỉ số:

- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- R-squared (R2 score)

Đoạn code minh họa việc xây dựng preprocessor và huấn luyện mô hình trong `training_pipeline.py`:

```
def build_preprocessor(num_cols, cat_cols):
    return ColumnTransformer([
        ('num', Pipeline([('imp', KNNImputer()), ('scale', StandardScaler())]),
        ↪  num_cols),
        ('cat', Pipeline([('imp', SimpleImputer(strategy='most_frequent')),
                          ('oh', OneHotEncoder(handle_unknown='ignore',
                          ↪  sparse_output=False))]), cat_cols) #
    ])

def train(X_train, X_test, y_train, y_test, prep, plot_dir):
    X_train_prep, X_test_prep = prep.fit_transform(X_train), prep.transform(X_test)
    model = GradientBoostingRegressor(random_state=42)
    param_grid = {
        'n_estimators': randint(100, 500),
        'learning_rate': uniform(0.01, 0.2),
        'max_depth': randint(3, 6),
```

```
        'min_samples_split': randint(2, 10),
        'min_samples_leaf': randint(1, 10),
        'subsample': uniform(0.7, 0.3)
    }
    search = RandomizedSearchCV(model, param_grid, n_iter=50, cv=5,
                                scoring='neg_root_mean_squared_error', n_jobs=-1,
                                random_state=42, verbose=1)
    search.fit(X_train_prep, y_train)
    evaluate(search.best_estimator_, X_test_prep, y_test,
    prep.get_feature_names_out(), plot_dir)
    return search.best_estimator_, prep
```

4.4.4 Quy trình huấn luyện tổng thể

Module `training_pipeline.py` khi được chạy sẽ thực hiện tuần tự các bước:

`load_data()`: Tải dữ liệu thống kê cầu thủ (Chương 1) và dữ liệu giá trị chuyển nhượng (Chương 4), sau đó hợp nhất chúng.

`split_and_prep_data()`: Chuẩn bị dữ liệu, áp dụng biến đổi log cho biến mục tiêu, tách thành tập huấn luyện/kiểm tra, xác định các cột số và hạng mục, xử lý sơ bộ các giá trị thiếu và loại bỏ cột có quá nhiều giá trị thiếu.

`build_preprocessor()`: Xây dựng đối tượng `ColumnTransformer` cho tiền xử lý.

`train()`: Huấn luyện mô hình, bao gồm áp dụng preprocessor, tìm kiếm siêu tham số tối ưu, huấn luyện mô hình cuối cùng và gọi hàm `evaluate()`.

`evaluate()`: Tính toán các chỉ số đánh giá (RMSE, MAE, R2) và tạo các biểu đồ trực quan hóa:

- Mức độ quan trọng của các đặc trưng (Feature Importance).
- Biểu đồ phân tán giữa giá trị dự đoán và giá trị thực tế (Predicted vs Actual).
- Biểu đồ phần dư (Residuals plot).

4.4.5 Kết quả và đánh giá

Phân tích các chỉ số đánh giá mô hình

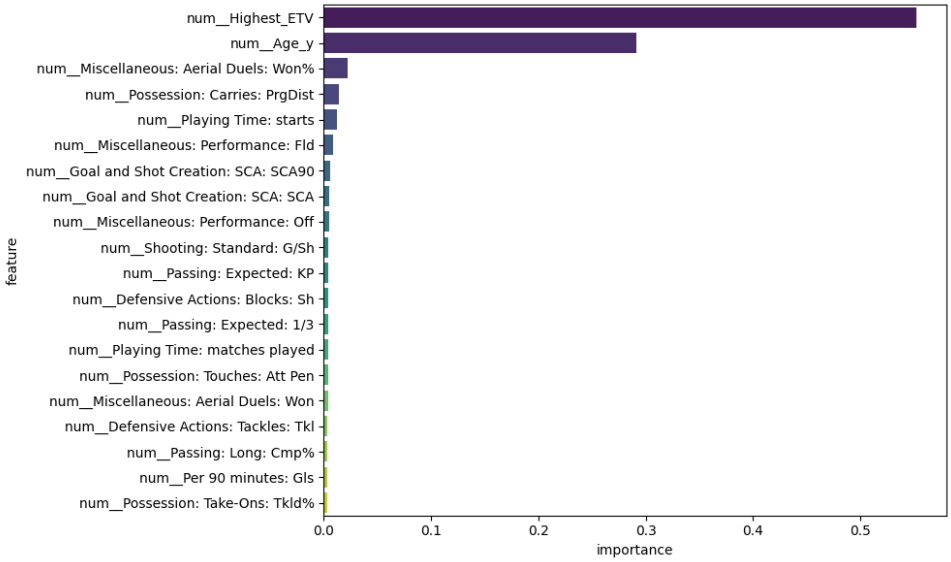
Mô hình ước tính giá trị cầu thủ đạt được các kết quả sau trên tập kiểm tra:

- RMSE (Root Mean Squared Error) là 7.06, cho thấy sai số trung bình giữa giá trị dự đoán và giá trị thực tế là khoảng 7.06 triệu EUR.
- MAE (Mean Absolute Error): 5.60. MAE đo lường sai số tuyệt đối trung bình giữa giá trị dự đoán và giá trị thực tế.
- Giá trị MAE thể hiện mô hình dự đoán sai lệch 5.60 triệu EUR so với giá trị thực.
- MAE ít bị ảnh hưởng bởi các giá trị ngoại lệ lớn hơn so với RMSE.
- R-squared (R2 score): 0.902. Chỉ số R2 cho biết tỷ lệ phương sai của biến mục tiêu (giá trị cầu thủ) mà mô hình có thể giải thích được.

- Giá trị 0.902 (hay 90.2%) là một kết quả rất tốt, cho thấy mô hình giải thích được hơn 90% sự biến động trong giá trị chuyển nhượng của cầu thủ dựa trên các đặc trưng đã chọn.

Nhìn chung, các chỉ số này cho thấy mô hình có độ chính xác khá cao trong việc ước tính giá trị cầu thủ. Mặc dù có sự chênh lệch nhất định (thể hiện qua RMSE và MAE), khả năng giải thích phần lớn sự biến thiên giá trị (R2) là một dấu hiệu tích cực.

Phân tích Biểu đồ Mức độ Quan trọng của Đặc trưng



Hình 4.2: Biểu đồ mức độ quan trọng

Biểu đồ mức độ quan trọng của đặc trưng trên cung cấp thông tin về những yếu tố nào có ảnh hưởng lớn nhất đến việc dự đoán giá trị cầu thủ của mô hình Gradient Boosting Regressor. Các đặc trưng quan trọng nhất:

- num_Highest_ETV: Giá trị chuyển nhượng ước tính (ETV) cao nhất từng được ghi nhận của cầu thủ.

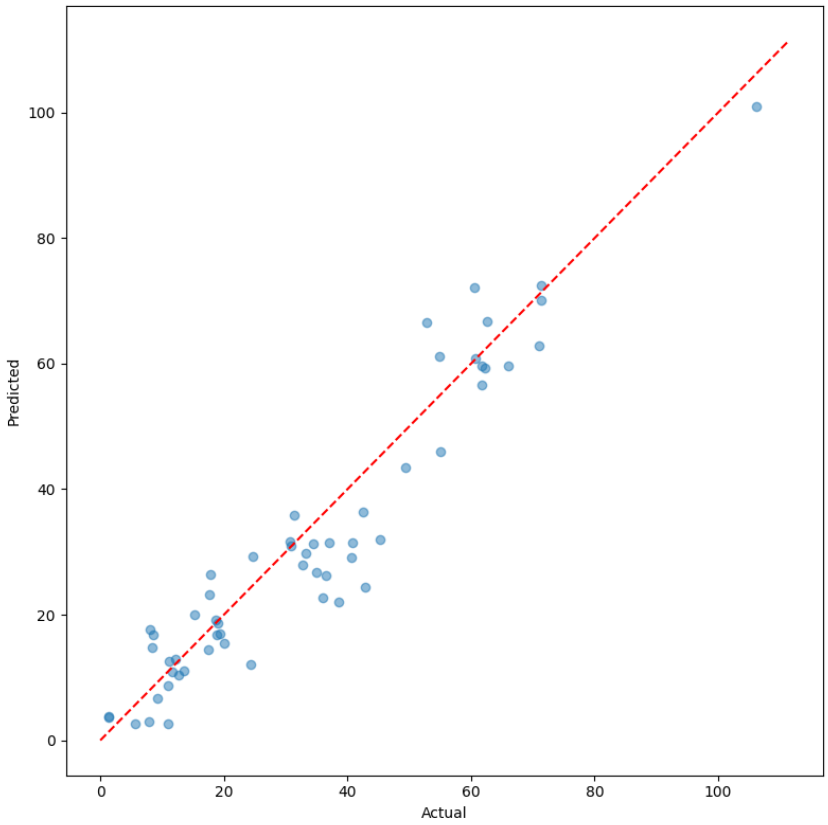
  Đây là đặc trưng có ảnh hưởng lớn nhất. Điều này hợp lý vì giá trị lịch sử thường là một chỉ báo mạnh mẽ cho giá trị hiện tại.

- num_Age_y: Tuổi của cầu thủ. Tuổi tác ảnh hưởng lớn đến tiềm năng phát triển, giá trị bán lại và đỉnh cao phong độ của cầu thủ, do đó nó là một yếu tố dự đoán quan trọng.

Các đặc trưng có ảnh hưởng đáng kể khác:

- num_Miscellaneous: Aerial Duels: Won% (Tỷ lệ không chiến thành công)
- num_Possession: Carries: PrgDist (Tổng quãng đường mang bóng tịnh tiến)
- num_Playing Time: starts (Số trận đá chính)

Phân tích Biểu đồ Giá trị Dự đoán và Thực tế

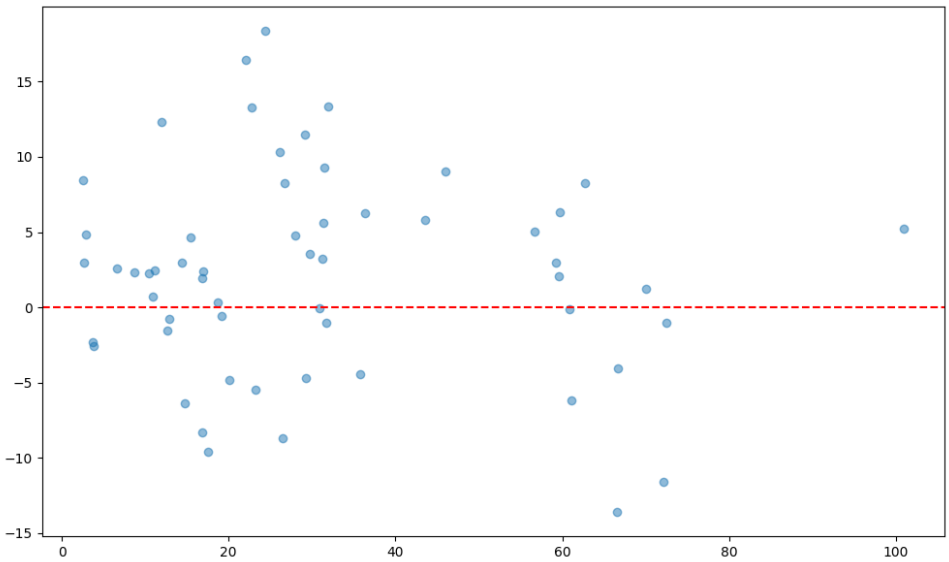


Hình 4.3: Biểu đồ giữa giá trị dự đoán và thực tế

Biểu đồ trên trực quan hóa mức độ khớp giữa giá trị chuyển nhượng dự đoán bởi mô hình và giá trị thực tế.

- Đánh giá chung: Các điểm dữ liệu có xu hướng tập trung quanh đường chéo y=x.
- Điều này cho thấy mô hình dự đoán khá tốt trên một phạm vi rộng các giá trị cầu thủ.
- Kết luận: Biểu đồ này củng cố kết quả R2 cao, cho thấy mô hình nắm bắt được xu hướng chung của dữ liệu.
- Tuy nhiên, cần lưu ý đến khả năng dự đoán kém chính xác hơn ở các mức giá trị cực đoan.

Phân tích Biểu đồ Phần dư



Hình 4.4: Dữ liệu phần dư phân phối

Biểu đồ phần dư trên thể hiện sự khác biệt (sai số) giữa giá trị dự đoán và giá trị thực tế cho mỗi điểm dữ liệu.

- Phân phối của phần dư: Lý tưởng nhất, phần dư nên được phân phối ngẫu nhiên xung quanh đường 0, không có khuôn mẫu rõ ràng.
- Trong Hình, phần lớn các điểm phần dư phân bố quanh đường 0.

Kết luận: Biểu đồ phần dư không cho thấy vấn đề nghiêm trọng nào với mô hình. Phần lớn sai số là nhỏ và phân bố ngẫu nhiên.

4.4.6 Đánh giá tổng thể phương pháp đề xuất

Dựa trên các phân tích trên, phương pháp đề xuất để ước tính giá trị cầu thủ khá hiệu quả và phù hợp với yêu cầu đề bài Chương 4.

- Thu thập dữ liệu: Quá trình thu thập dữ liệu từ footballtransfers.com và lọc theo thời gian thi đấu (>900 phút) đã cung cấp bộ dữ liệu mục tiêu cần thiết.
- Lựa chọn đặc trưng: Việc kết hợp các chỉ số thống kê hiệu suất từ Chương 1, thông tin cơ bản của cầu thủ (tuổi, vị trí) và giá trị chuyển nhượng lịch sử (ETV cao nhất) là một chiến lược lựa chọn đặc trưng toàn diện.
- Kết quả phân tích mức độ quan trọng của đặc trưng đã xác nhận sự đóng góp của các loại thông tin này.
- Lựa chọn mô hình: Mô hình Gradient Boosting Regressor, sau khi được tối ưu hóa siêu tham số, đã cho thấy khả năng dự đoán tốt với R2 đạt 0.902.
- Khả năng xử lý các mối quan hệ phi tuyến và tương tác giữa các đặc trưng của mô hình này là phù hợp với sự phức tạp của việc định giá cầu thủ.

Tài liệu tham khảo

[1] Codecademy Team. Python web scraping using selenium and beautiful soup: A step-by-step tutorial, 3 2024.

[2] GeeksforGeeks. Scrape content from dynamic websites, 1 2025.

[3] Gowsic K, Mugunthan S, Sakthivel Logavaseekarapakther, Puviyarasu A, and Mohammed Farook R. Enhanced unsupervised k-means clustering algorithm. *ShodhKosh: Journal of Visual and Performing Arts*, 5(1):1141–1150, 1 2024.

[4] Carson Kai-Sang Leung, Anh Le Tran, Christopher Leckie, and Kotagiri Ramamohanarao. Comparing machine learning and ensemble learning in the field of football. *ResearchGate*, 2019.

[5] Łukasz Marciniak, Tomasz Marciniak, and Piotr Szwajkowski. Football player value prediction: Comparing machine learning models with cross-validation. *ResearchGate*, 2024.

[6] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate analysis*. Academic Press, London; New York, 1979.

[7] Chantha Wongoutong. The impact of neglecting feature scaling in k-means clustering. *PLOS ONE*, 19(12):e0310839, 2024.