BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI BÁO CÁO

BÀI TẬP LỚN: PYTHON

Giảng viên: Kim Ngọc Bách

Lớp: 06 – D23CQCE06 - B

Sinh viên: Vũ Đình Hiếu B23DCCE036

Hà Nội, Tháng 5/2025

Mục lục

1 Lời mở đầu **3**

2 Chương 1: Thu thập và xử lí dữ liệu **3**

2.1 Tổng quan về mã code và giải thích ý tưởng chung 3

2.2 Lấy nội dung của dữ liệu theo các thẻ 3

2.3 Trích xuất dữ liệu theo từng hàng ứng với các giá trị thẻ HTML 3

2.4 Thu thập dữ liệu từ bảng thống kê trên website 3

2.5 Định nghĩa danh sách thuộc tính tạo danh sách cặp 3

2.6 Phân chia các dataframe thành các trường dữ liệu từ các url và hợp nhất các dataframe 3

2.7 Xây dựng tiêu đề cho các trường dữ liệu 3

3 Chương 2: Xử lý dữ liệu và thống kê dữ liệu **3**

3.1 Tổng quan về Code - Problem 1 3

3.2 Tổng quan về Code - Problem 2 3

3.3 Tổng quan về Code - Problem 3 3

4 Chương 3: Phân cụm dữ liệu **3**

5 Chương 4: Thu thập dữ liệu và xây dựng mô hình dự đoán giá trị chuyển nhượng **3**

1 Lời mở đầu

Bóng đá, môn thể thao vua, không chỉ là cuộc chiến trên sân cỏ mà còn là một lĩnh vực mà dữ liệu và công nghệ đang ngày càng khẳng định vai trò quan trọng. Trong kỷ nguyên số, các chỉ số thống kê về hiệu suất cầu thủ đã trở thành nền tảng để các câu lạc bộ, huấn luyện viên, và nhà phân tích đưa ra những quyết định chiến lược, từ việc xây dựng đội hình đến các thương vụ chuyển nhượng trị giá hàng triệu đô-la. Với mục tiêu ứng dụng ngôn ngữ lập trình Python – một trong những công cụ mạnh mẽ nhất trong phân tích dữ liệu, bài tập lớn này được thực hiện để khám phá và khai thác dữ liệu liên quan đến các chỉ số thống kê của các cầu thủ tại Giải Ngoại hạng Anh (Premier League), giải đấu được xem là đỉnh cao của bóng đá thế giới.

Nội dung bài tập lớn tập trung vào ba giai đoạn chính: thu thập, xử lý và phân tích dữ liệu. Đầu tiên, dữ liệu về các chỉ số thống kê của cầu thủ, bao gồm số bàn thắng, kiến tạo, tỷ lệ chuyền bóng chính xác, số lần tắc bóng, và nhiều chỉ số khác, sẽ được thu thập từ các nguồn đáng tin cậy như trang web chính thức của Premier League, các nền tảng thống kê thể thao, hoặc các bộ dữ liệu công khai. Quá trình xử lý dữ liệu bao gồm làm sạch dữ liệu, xử lý giá trị thiếu, chuẩn hóa định dạng, và tính toán các chỉ số tổng hợp để đảm bảo dữ liệu sẵn sàng cho các bước phân tích tiếp theo. Các thư viện Python như Pandas và NumPy sẽ được sử dụng để thực hiện các tác vụ này một cách hiệu quả.

Sau khi dữ liệu được chuẩn bị, bài tập áp dụng kỹ thuật phân cụm (clustering) để phân loại cầu thủ thành các nhóm dựa trên đặc điểm hiệu suất, chẳng hạn như cầu thủ tấn công, cầu thủ phòng ngự, hoặc tiền vệ toàn diện. Các thuật toán như K-Means sẽ được triển khai thông qua thư viện Scikit-learn để thực hiện nhiệm vụ này. Kết quả phân cụm không chỉ giúp hiểu rõ hơn về vai trò và phong cách thi đấu của cầu thủ mà còn hỗ trợ các câu lạc bộ trong việc xác định những mục tiêu chuyển nhượng phù hợp. Hơn nữa, bài tập xây dựng một mô hình dự đoán giá trị chuyển nhượng của cầu thủ dựa trên các chỉ số thống kê, độ tuổi, vị trí thi đấu, và thành tích thi đấu.

Thông qua việc sử dụng các công cụ trực quan hóa dữ liệu như Matplotlib và Seaborn, bài tập cũng trình bày các kết quả phân tích dưới dạng biểu đồ và đồ thị, giúp minh họa rõ ràng các xu hướng và mối quan hệ trong dữ liệu. Bài tập lớn này không chỉ là cơ hội để rèn luyện kỹ năng lập trình Python và phân tích dữ liệu mà còn thể hiện tiềm năng ứng dụng công nghệ vào việc nâng cao hiệu quả quản lý và ra quyết định trong lĩnh vực bóng đá. Với tính thực tiễn và ý nghĩa học thuật, bài tập mang đến một góc nhìn mới về cách dữ liệu có thể thay đổi cách chúng ta tiếp cận môn thể thao vua.

- **Chương 1**: Thu thập và xử lý dữ liệu
  Trong chương này chúng ta sẽ đi vào phân tích dữ liệu và thu thập dữ liệu các chỉ số thống kê của các cầu thủ bóng đá theo yêu cầu: tất cả cầu thủ bóng đá phải chơi trên 90 phút trong mùa giải 2024 - 2025 của giải English Premier League.

- **Chương 2**: Xử lý dữ liệu và thống kê dữ liệu
  Chương này sẽ đi vào xử lý dữ liệu và thống kê các chỉ số theo yêu cầu: chọn ra 3 cầu thủ có chỉ số cao nhất và 3 cầu thủ có chỉ số thấp nhất cho mỗi cột dữ liệu. Tính toán giá trị trung bình, độ lệch chuẩn cho mỗi giá trị thống kê của tất cả cầu thủ cho mỗi đội, trực quan hoá dữ liệu bằng biểu đồ histogram với mỗi thuộc tính của tất cả cầu thủ và mỗi đội.

- **Chương 3**: Dự đoán và trực quan hoá dữ liệu
  Chương này sẽ đi vào xây dựng mô hình và thuật toán để phân cụm các cầu thủ

(phân loại), sử dụng thuật toán K-means để phân loại các cầu thủ và PCA để giảm chiều dữ liệu rồi trực quan hoá dữ liệu.

- **Chương 4**: Thu thập dữ liệu và xây dựng mô hình để dự đoán giá trị chuyển nhượng Thu thập dữ liệu về giá trị chuyển nhượng của các cầu thủ trong mùa giải 2024 - 2025 và lấy ra các cầu thủ chơi bóng trên 900 phút. Xây dựng model để dự đoán giá trị chuyển nhượng.

2 Chương 1: Thu thập và xử lý dữ liệu

Trong chương này chúng ta sẽ đi vào phân tích dữ liệu và thu thập dữ liệu các chỉ số thống kê của các cầu thủ bóng đá theo yêu cầu: tất cả cầu thủ bóng đá phải chơi trên 90 phút trong mùa giải 2024 - 2025 của giải English Premier League.

2.1 Tổng quan về mã code và giải thích ý tưởng chung

Mã này thực hiện việc thu thập dữ liệu thống kê bóng đá từ trang web fbref.com, cụ thể là các số liệu về giải Premier League, sau đó tổ chức và lưu dữ liệu vào file CSV. Ý tưởng chính là sử dụng Selenium để tải trang web động, BeautifulSoup để phân tích HTML, và Pandas để xử lý dữ liệu. Mã được thiết kế để cào dữ liệu từ nhiều bảng khác nhau (thống kê tiêu chuẩn, thủ môn, sút bóng, chuyền bóng, phòng ngự, sở hữu bóng, v.v.), lọc các cầu thủ chơi ít nhất 1 trận (90 phút), và hợp nhất dữ liệu dựa trên các cột chung như tên cầu thủ, đội, vị trí.

Quy trình hoạt động:

- Hàm `create_attributes` tạo danh sách các thuộc tính cần cào cùng với cách trích xuất dữ liệu (mặc định lấy văn bản hoặc dùng hàm tùy chỉnh cho các trường đặc biệt như quốc tịch, năm sinh).
- Hàm `extract_data` trích xuất dữ liệu từ mỗi hàng (tr) của bảng HTML dựa trên các thuộc tính đã định nghĩa.
- Hàm `scrape_table` tải trang web, phân tích bảng HTML, lọc các hàng thỏa mãn điều kiện (ví dụ: thời gian thi đấu tối thiểu, vị trí thủ môn nếu cần), và trả về DataFrame.
- Dữ liệu từ các bảng được cào riêng biệt (thống kê tiêu chuẩn, thủ môn, v.v.), sau đó hợp nhất bằng phương thức `merge` dựa trên các cột chung.
- Kết quả được sắp xếp theo tên cầu thủ, thêm cột số thứ tự (No.), xử lý giá trị NaN bằng cách thay thế thành 'N/A', và tổ chức thành MultiIndex để phân loại cột theo nhóm (ví dụ: Goalkeeping, Shooting, Passing).
- Cuối cùng, dữ liệu được lưu vào file `results.csv` và trình duyệt Selenium được đóng.

Mục đích: Tạo ra một tập dữ liệu chi tiết, có cấu trúc rõ ràng về hiệu suất của các cầu thủ Premier League, phù hợp cho phân tích hoặc báo cáo.

```python
from bs4 import BeautifulSoup
import time
from selenium import webdriver
import pandas as pd

def create_attributes(data_stats, special_extractors=None):
    special_extractors = special_extractors or {}
    return [(stat, special_extractors.get(stat, lambda td: td.get_text(strip=True))) for stat in data_stats]

def extract_data(tr, attributes):
    row_data = {}
    for data_stat, extractor in attributes:
        td = tr.find('td', {'data-stat': data_stat})
        row_data[data_stat] = extractor(td) if td else None
    return row_data

def scrape_table(driver, url, attributes, is_goalkeeper=False):
    driver.get(url)
    time.sleep(0.5)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    table = soup.find('table', {'class': 'min_width sortable stats_table shade_zero now_sortable sticky_table eq2 re2 le2'})
    if not table:
        return pd.DataFrame()

    tbody = table.find('tbody')
    data = []
    for tr in tbody.find_all('tr'):
        if 'thead' in tr.get('class', []):
            continue
        minutes_90s = tr.find('td', {'data-stat': 'minutes_90s'}).get_text(strip=True)
        if float(minutes_90s) < 1.0:
            continue
        if is_goalkeeper and 'GK' not in tr.find('td', {'data-stat': 'position'}).get_text(strip=True):
            continue
        row_data = extract_data(tr, attributes)
        data.append(row_data)

    return pd.DataFrame(data)

driver = webdriver.Chrome()

standard_stats = [
    'player', 'nationality', 'team', 'position', 'birth_year', 'games', 'games_starts', 'minutes',
    'goals', 'assists', 'cards_yellow', 'cards_red', 'xg', 'xg_assist', 'progressive_carries',
    'progressive_passes', 'progressive_passes_received', 'goals_per90', 'assists_per90',
    'xg_per90', 'xg_assist_per90'
```

```
]
standard_attributes = create_attributes(standard_stats, {
    'nationality': lambda td: td.find('a').find('span').contents[-1].strip(),
    'birth_year': lambda td: str(2025 - int(td.get_text(strip=True)))
})

goalkeeping_attributes = create_attributes([
    'player', 'team', 'position', 'gk_goals_against_per90', 'gk_save_pct', 'gk_clean_sheets_pct', 'gk_pens_save_pct'
])

shooting_attributes = create_attributes([
    'player', 'team', 'position', 'shots_on_target_pct', 'shots_on_target_per90', 'goals_per_shot', 'average_shot_distance'
])

passing_attributes = create_attributes([
    'player', 'team', 'position', 'passes_completed', 'passes_pct', 'passes_total_distance', 'passes_pct_short',
    'passes_pct_medium', 'passes_pct_long', 'assisted_shots', 'passes_into_final_third',
    'passes_into_penalty_area', 'crosses_into_penalty_area', 'progressive_passes'
])

gca_attributes = create_attributes(['player', 'team', 'position', 'sca', 'sca_per90', 'gca', 'gca_per90'])

defense_attributes = create_attributes([
    'player', 'team', 'position', 'tackles', 'tackles_won', 'challenges', 'challenges_lost', 'blocks', 'blocked_shots',
    'blocked_passes', 'interceptions'
])

possession_attributes = create_attributes([
    'player', 'team', 'position', 'touches', 'touches_def_pen_area', 'touches_def_3rd', 'touches_mid_3rd', 'touches_att_3rd',
    'touches_att_pen_area', 'take_ons', 'take_ons_won', 'take_ons_tackled', 'carries',
    'carries_progressive_distance', 'progressive_carries', 'carries_into_final_third',
    'carries_into_penalty_area', 'miscontrols', 'dispossessed', 'passes_received',
    'progressive_passes_received'
])

misc_attributes = create_attributes([
    'player', 'team', 'position', 'fouls', 'fouled', 'offsides', 'crosses', 'ball_recoveries', 'aerials_won',
    'aerials_lost', 'aerials_won_pct'
])

```

```python
df_standard = scrape_table(driver, 'https://fbref.com/en/comps/9/stats/Premier-League-Stats', standard_attributes)
df_goalkeeping = scrape_table(driver, 'https://fbref.com/en/comps/9/keepers/Premier-League-Stats', goalkeeping_attributes, is_goalkeeper=True)
df_shooting = scrape_table(driver, 'https://fbref.com/en/comps/9/shooting/Premier-League-Stats', shooting_attributes)
df_passing = scrape_table(driver, 'https://fbref.com/en/comps/9/passing/Premier-League-Stats', passing_attributes)
df_gca = scrape_table(driver, 'https://fbref.com/en/comps/9/gca/Premier-League-Stats', gca_attributes)
df_defense = scrape_table(driver, 'https://fbref.com/en/comps/9/defense/Premier-League-Stats', defense_attributes)
df_possession = scrape_table(driver, 'https://fbref.com/en/comps/9/possession/Premier-League-Stats', possession_attributes)
df_misc = scrape_table(driver, 'https://fbref.com/en/comps/9/misc/Premier-League-Stats', misc_attributes)

df = df_standard
for df_temp in [df_goalkeeping, df_shooting, 'df_passing', df_gca, df_defense, df_possession, df_misc]:
    df = df.merge(df_temp, on=['player', 'team', 'position'], how='left')

df = df.sort_values(by='player')

# Adding No. column and keeping player column
df.insert(0, 'No.', range(len(df)))

# Creating a copy and handling NaN values
tmp = df.copy()
pd.set_option('future.no_silent_downcasting', True)
tmp = tmp.fillna('N/A')
tmp = tmp.astype(str)
tmp = tmp.replace(['nan', 'NaN', '', None], 'N/A')

# Defining MultiIndex for columns, adding Player column after No.
columns = [('', '', 'No.'), ('', '', 'Player')] + [
    ('', 'Nation', 'nationality'), ('', 'Team', 'team'), ('', 'Position', 'position'),
    ('', 'Age', 'birth_year'), ('', 'Playing Time', 'games'), ('', '', 'games_starts'), ('', '', 'minutes'),
    ('', 'Performance', 'goals'), ('', '', 'assists'), ('', '', 'cards_yellow'), ('', '', 'cards_red'),
    ('', 'Expected', 'xg'), ('', '', 'xg_assist'), ('', 'Progression', 'progressive_carries'),
    ('', '', 'progressive_passes'), ('', '',
```

```
        'progressive_passes_received'), ('', 'Per 90 minutes',
        'goals_per90'),
    ('', '', 'assists_per90'), ('', '', 'xg_per90'), ('', '',
        'xg_assist_per90'),
    ('Goalkeeping', 'Performance', 'gk_goals_against_per90'), ('', '',
        'gk_save_pct'), ('', '', 'gk_clean_sheets_pct'),
    ('', 'Penalty Kicks', 'gk_pens_save_pct'), ('Shooting', 'Standard',
        'shots_on_target_pct'),
    ('', '', 'shots_on_target_per90'), ('', '', 'goals_per_shot'), ('',
        '', 'average_shot_distance'),
    ('Passing', 'Total', 'passes_completed'), ('', '', 'passes_pct'),
        ('', '', 'passes_total_distance'),
    ('', 'Short', 'passes_pct_short'), ('', 'Medium',
        'passes_pct_medium'), ('', 'Long', 'passes_pct_long'),
    ('', 'Expected', 'assisted_shots'), ('', '',
        'passes_into_final_third'), ('', '', 'passes_into_penalty_area'),
    ('', '', 'crosses_into_penalty_area'), ('', '',
        'progressive_passes'), ('Goal and Shot Creation', 'SCA', 'sca'),
    ('', '', 'sca_per90'), ('', 'GCA', 'gca'), ('', '', 'gca_per90'),
        ('Defensive Actions', 'Tackles', 'tackles'),
    ('', '', 'tackles_won'), ('', 'Challenges', 'challenges'), ('', '',
        'challenges_lost'),
    ('', 'Blocks', 'blocks'), ('', '', 'blocked_shots'), ('', '',
        'blocked_passes'), ('', '', 'interceptions'),
    ('Possession', 'Touches', 'touches'), ('', '',
        'touches_def_pen_area'), ('', '', 'touches_def_3rd'),
    ('', '', 'touches_mid_3rd'), ('', '', 'touches_att_3rd'), ('', '',
        'touches_att_pen_area'),
    ('', 'Take-Ons', 'take_ons'), ('', '', 'take_ons_won'), ('', '',
        'take_ons_tackled'),
    ('', 'Carries', 'carries'), ('', '',
        'carries_progressive_distance'), ('', '', 'progressive_carries'),
    ('', '', 'carries_into_final_third'), ('', '',
        'carries_into_penalty_area'), ('', '', 'miscontrols'),
    ('', '', 'dispossessed'), ('', 'Receiving', 'passes_received'),
        ('', '', 'progressive_passes_received'),
    ('Miscellaneous Stats', 'Performance', 'fouls'), ('', '',
        'fouled'), ('', '', 'offsides'), ('', '', 'crosses'),
    ('', '', 'ball_recoveries'), ('', 'Aerial Duels', 'aerials_won'),
        ('', '', 'aerials_lost'),
    ('', '', 'aerials_won_pct')
]

multi_cols = pd.MultiIndex.from_tuples(columns)
tmp.columns = multi_cols

# Saving to CSV file
tmp.to_csv('results.csv', index=False)

driver.quit()
```

2.2 Lấy nội dung của dữ liệu theo các thẻ (div, tag, class)

Code:

```
def create_attributes(data_stats, special_extractors=None):
    special_extractors = special_extractors or {}
    return [(stat, special_extractors.get(stat, lambda td:
        td.get_text(strip=True))) for stat in data_stats]
```

Mục đích

Hàm tạo danh sách các cặp, trong đó mỗi cặp gồm tên một thuộc tính (ví dụ: `player`, `goals`) và một hàm trích xuất dữ liệu tương ứng. Các hàm trích xuất này được dùng để lấy dữ liệu từ thẻ HTML (thường là `<td>`) trong bảng dữ liệu, hỗ trợ xử lý linh hoạt các trường dữ liệu khác nhau, bao gồm các trường cần logic trích xuất đặc biệt (như quốc tịch hoặc năm sinh).

Tham số đầu vào

- `data_stats`: Danh sách các chuỗi, mỗi chuỗi là tên của một thuộc tính cần trích xuất (ví dụ: `['player', 'goals', 'assists']`). Đây là tham số bắt buộc, xác định các cột dữ liệu sẽ thu thập.
- `special_extractors`: Từ điển tùy chọn, trong đó khóa là tên thuộc tính (chuỗi), giá trị là hàm lambda hoặc hàm tùy chỉnh để trích xuất dữ liệu. Mặc định là `None`.

Cơ chế hoạt động

- **Khởi tạo** `special_extractors`: Dòng `special_extractors = special_extractors or {}` đảm bảo nếu tham số `special_extractors` là `None`, nó sẽ được thay bằng từ điển rỗng, tránh lỗi khi truy cập từ điển.
- **Tạo danh sách cặp thuộc tính-hàm trích xuất**: Hàm sử dụng *list comprehension* để tạo danh sách các tuple:

  ```
  [(stat, special_extractors.get(stat, lambda td:
      td.get_text(strip=True))) for stat in data_stats]
  ```

  - Với mỗi `stat` trong `data_stats`, hàm kiểm tra xem `special_extractors` có hàm trích xuất tùy chỉnh cho `stat` không bằng phương thức `get(stat, default)`.
  - Nếu có hàm tùy chỉnh (ví dụ: `special_extractors['nationality']`), hàm đó được dùng.
  - Nếu không, hàm mặc định `lambda td: td.get_text(strip=True)` được sử dụng, lấy nội dung văn bản từ thẻ `<td>` và loại bỏ khoảng trắng thừa.

Kết quả trả về

Hàm trả về danh sách các tuple, mỗi tuple chứa:

- Một chuỗi (tên thuộc tính, ví dụ: `player`).
- Một hàm (lambda hoặc tùy chỉnh) nhận đầu vào là đối tượng BeautifulSoup (thường là thẻ `<td>`) và trả về dữ liệu đã trích xuất (thường là chuỗi hoặc giá trị đã xử lý).

2.3 Trích xuất dữ liệu theo "từng" hàng ứng với các giá trị của thẻ HTML

Code:

```python
def extract_data(tr, attributes):
    row_data = {}
    for data_stat, extractor in attributes:
        td = tr.find('td', {'data-stat': data_stat})
        row_data[data_stat] = extractor(td) if td else None
    return row_data
```

Mục đích

Hàm trích xuất dữ liệu từ một hàng (thẻ `<tr>`) của bảng HTML, lấy giá trị của các cột được xác định bởi danh sách `attributes`. Mỗi thuộc tính đi kèm một hàm trích xuất để xử lý dữ liệu từ thẻ `<td>`. Hàm trả về một từ điển chứa các cặp khóa-giá trị, trong đó khóa là tên thuộc tính và giá trị là dữ liệu đã trích xuất, nhằm thu thập dữ liệu có cấu trúc để tạo DataFrame hoặc xử lý tiếp.

Tham số đầu vào

- `tr`: Đối tượng BeautifulSoup đại diện cho thẻ `<tr>` (hàng) trong bảng HTML, chứa các thẻ `<td>` với dữ liệu cần trích xuất.
- `attributes`: Danh sách các tuple, mỗi tuple chứa:
  - `data_stat`: Chuỗi, tên thuộc tính cần trích xuất (ví dụ: `player`, `goals`).
  - `extractor`: Hàm nhận đầu vào là thẻ `<td>` (hoặc `None`) và trả về dữ liệu đã xử lý (thường là chuỗi hoặc giá trị khác).

Cơ chế hoạt động

- **Khởi tạo từ điển rỗng**: Hàm tạo từ điển `row_data` để lưu các cặp khóa-giá trị (tên thuộc tính và giá trị trích xuất).
- **Duyệt qua danh sách** `attributes`: Với mỗi tuple `(data_stat, extractor)`:
  - Hàm dùng `tr.find('td', {'data-stat': data_stat})` để tìm thẻ `<td>` có thuộc tính `data-stat` khớp với `data_stat`.

- Nếu tìm thấy thẻ (`td` không phải `None`), gọi `extractor(td)` để trích xuất dữ liệu và lưu vào `row_data[data_stat]`.
- Nếu không tìm thấy thẻ (`td` là `None`), gán `None` cho `row_data[data_stat]`.

- **Trả về từ điển**: Sau khi xử lý tất cả thuộc tính, hàm trả về `row_data`.

Kết quả trả về

Hàm trả về một từ điển (`row_data`), trong đó:

- Khóa: Chuỗi `data_stat` (tên thuộc tính, ví dụ: `player`, `goals`).
- Giá trị: Kết quả của hàm `extractor` (thường là chuỗi, số, hoặc `None` nếu không tìm thấy thẻ `<td>`).

Ví dụ:

```
tr = BeautifulSoup('<tr><td data-stat="player">Messi</td><td data-stat="goals">10</td></tr>', 'html.parser')
attributes = [('player', lambda td: td.get_text(strip=True)), ('goals', lambda td: int(td.get_text(strip=True)))]
result = extract_data(tr, attributes)
```

Kết quả:

```
{'player': 'Messi', 'goals': 10}
```

Ứng dụng

Hàm được dùng trong hàm `scrape_table` để trích xuất dữ liệu từ mỗi hàng của bảng HTML, sau đó các từ điển được gom thành DataFrame bằng Pandas. Hàm giúp tự động hóa việc thu thập dữ liệu từ các bảng thống kê bóng đá trên trang fbref.com, đảm bảo dữ liệu chính xác và có cấu trúc.

2.4 Thu thập dữ liệu từ bảng thống kê trên website

Hàm truy cập trang web, trích xuất dữ liệu từ các hàng của bảng, lọc dựa trên tiêu chí cụ thể, và trả về dữ liệu dưới dạng DataFrame.

Code:

```
def scrape_table(driver, url, attributes, is_goalkeeper=False):
    driver.get(url)
    time.sleep(0.5)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    table = soup.find('table', {'class': 'min_width sortable stats_table shade_zero now_sortable sticky_table eq2 re2 le2'})
    if not table:
        return pd.DataFrame()

```

```
    tbody = table.find('tbody')
    data = []
    for tr in tbody.find_all('tr'):
        if 'thead' in tr.get('class', []):
            continue
        minutes_90s = tr.find('td', {'data-stat':
            'minutes_90s'}).get_text(strip=True)
        if float(minutes_90s) < 1.0:
            continue
        if is_goalkeeper and 'GK' not in tr.find('td', {'data-stat':
            'position'}).get_text(strip=True):
            continue
        row_data = extract_data(tr, attributes)
        data.append(row_data)

    return pd.DataFrame(data)
```

Mục đích

Hàm truy cập trang web chứa bảng thống kê, trích xuất dữ liệu từ các hàng dựa trên danh sách thuộc tính, và trả về dữ liệu dưới dạng DataFrame. Hàm lọc các cầu thủ có thời gian thi đấu ít nhất 1 trận (90 phút) và, nếu được chỉ định, chỉ lấy dữ liệu của thủ môn, nhằm thu thập dữ liệu thống kê bóng đá (như bàn thắng, kiến tạo, cứu thua) để phân tích hoặc báo cáo.

Tham số đầu vào

- `driver`: Đối tượng trình duyệt của Selenium (ví dụ: `webdriver.Chrome()`), dùng để tải trang web động.
- `url`: Chuỗi, địa chỉ URL của trang chứa bảng dữ liệu (ví dụ: `https://fbref.com/en/comps/9/stats/Premier-League-Stats`).
- `attributes`: Danh sách các tuple, mỗi tuple chứa:
  - `data_stat`: Chuỗi, tên thuộc tính (ví dụ: `player`, `goals`).
  - `extractor`: Hàm nhận thẻ `<td>` (hoặc `None`) và trả về dữ liệu đã xử lý.
- `is_goalkeeper`: Boolean, mặc định `False`. Nếu `True`, chỉ xử lý hàng của cầu thủ có vị trí thủ môn (chứa `GK` trong cột vị trí).

Cơ chế hoạt động

- **Tải trang web**: Gọi `driver.get(url)` để tải trang, tạm dừng 0.5 giây (`time.sleep(0.5)`) để đảm bảo trang tải hoàn toàn.
- **Phân tích HTML**: Dùng `BeautifulSoup` để phân tích mã nguồn HTML (`driver.page_source`) với trình phân tích `html.parser`.
- **Tìm bảng**: Tìm thẻ `<table>` có lớp CSS `min_width sortable stats_table ...`. Nếu không tìm thấy, trả về DataFrame rỗng (`pd.DataFrame()`).

- **Trích xuất dữ liệu từ các hàng**:
  - Lấy thẻ `<tbody>` chứa các hàng dữ liệu.
  - Duyệt qua tất cả thẻ `<tr>` trong `<tbody>`:
    * Bỏ qua hàng có lớp `'thead'` (thường là tiêu đề).
    * Kiểm tra cột `minutes_90s` (thời gian thi đấu tính bằng số trận 90 phút). Nếu nhỏ hơn 1.0, bỏ qua.
    * Nếu `is_goalkeeper` là `True`, kiểm tra cột `position`. Nếu không chứa `GK`, bỏ qua.
    * Với hàng thỏa mãn, gọi `extract_data(tr, attributes)` để trích xuất dữ liệu và thêm vào danh sách `data`.
- **Tạo DataFrame**: Chuyển danh sách `data` thành DataFrame và trả về.

Kết quả trả về

Hàm trả về DataFrame của Pandas, trong đó:

- Mỗi hàng là dữ liệu của một cầu thủ, trích xuất từ thẻ `<tr>`.
- Mỗi cột tương ứng với một thuộc tính trong `attributes` (ví dụ: `player`, `goals`).
- Nếu không tìm thấy bảng hoặc không có hàng thỏa mãn, trả về DataFrame rỗng.

Ví dụ:

```
driver = webdriver.Chrome()
url = 'https://fbref.com/en/comps/9/stats/Premier-League-Stats'
attributes = [('player', lambda td: td.get_text(strip=True)), ('goals',
    lambda td: int(td.get_text(strip=True)))]
result = scrape_table(driver, url, attributes)
```

Kết quả có thể là:

```
   player  goals
0  Messi      10
1  Ronaldo     8
```

Ứng dụng

Hàm là cốt lõi trong quy trình cào dữ liệu từ các bảng thống kê bóng đá trên fbref.com. Nó được gọi nhiều lần với các URL và danh sách thuộc tính khác nhau (bảng thống kê tiêu chuẩn, thủ môn, chuyền bóng) để thu thập dữ liệu toàn diện, sau đó các DataFrame được hợp nhất để tạo tập dữ liệu tổng hợp.

2.5 Định nghĩa danh sách thuộc tính và tạo danh sách cặp

Đoạn mã dưới đây định nghĩa các danh sách thuộc tính và gọi hàm `create_attributes` để tạo danh sách các cặp (thuộc tính, hàm trích xuất) nhằm thu thập dữ liệu thống kê bóng đá từ bảng HTML trên trang fbref.com. Các danh sách thuộc tính đại diện cho các loại thống kê khác nhau (tiêu chuẩn, thủ môn, sút bóng, chuyền bóng, tạo cơ hội, phòng ngự, sở hữu bóng, và thống kê khác), và mỗi danh sách được dùng để trích xuất dữ liệu từ các bảng tương ứng.

Đoạn mã

```
standard_stats = [
    'player', 'nationality', 'team', 'position', 'birth_year', 'games',
        'games_starts', 'minutes',
    'goals', 'assists', 'cards_yellow', 'cards_red', 'xg', 'xg_assist',
        'progressive_carries',
    'progressive_passes', 'progressive_passes_received', 'goals_per90',
        'assists_per90',
    'xg_per90', 'xg_assist_per90'
]
standard_attributes = create_attributes(standard_stats, {
    'nationality': lambda td:
        td.find('a').find('span').contents[-1].strip(),
    'birth_year': lambda td: str(2025 - int(td.get_text(strip=True)))
})

goalkeeping_attributes = create_attributes([
    'player', 'team', 'position', 'gk_goals_against_per90',
        'gk_save_pct', 'gk_clean_sheets_pct', 'gk_pens_save_pct'
])

shooting_attributes = create_attributes([
    'player', 'team', 'position', 'shots_on_target_pct',
        'shots_on_target_per90', 'goals_per_shot',
        'average_shot_distance'
])

passing_attributes = create_attributes([
    'player', 'team', 'position', 'passes_completed', 'passes_pct',
        'passes_total_distance', 'passes_pct_short',
    'passes_pct_medium', 'passes_pct_long', 'assisted_shots',
        'passes_into_final_third',
    'passes_into_penalty_area', 'crosses_into_penalty_area',
        'progressive_passes'
])

gca_attributes = create_attributes(['player', 'team', 'position',
    'sca', 'sca_per90', 'gca', 'gca_per90'])

defense_attributes = create_attributes([
    'player', 'team', 'position', 'tackles', 'tackles_won',
        'challenges', 'challenges_lost', 'blocks', 'blocked_shots',
```

```
    'blocked_passes', 'interceptions'
])

possession_attributes = create_attributes([
    'player', 'team', 'position', 'touches', 'touches_def_pen_area',
        'touches_def_3rd', 'touches_mid_3rd', 'touches_att_3rd',
    'touches_att_pen_area', 'take_ons', 'take_ons_won',
        'take_ons_tackled', 'carries',
    'carries_progressive_distance', 'progressive_carries',
        'carries_into_final_third',
    'carries_into_penalty_area', 'miscontrols', 'dispossessed',
        'passes_received',
    'progressive_passes_received'
])

misc_attributes = create_attributes([
    'player', 'team', 'position', 'fouls', 'fouled', 'offsides',
        'crosses', 'ball_recoveries', 'aerials_won',
    'aerials_lost', 'aerials_won_pct'
])
```

Vai trò của các trường

Đoạn mã định nghĩa nhiều danh sách thuộc tính, mỗi danh sách tương ứng với một loại thống kê bóng đá từ các bảng khác nhau trên trang fbref.com. Các danh sách này được truyền vào hàm `create_attributes` để tạo cặp (thuộc tính, hàm trích xuất), giúp trích xuất dữ liệu từ thẻ `<td>` trong bảng HTML. Dưới đây là vai trò của từng danh sách thuộc tính:

- `standard_stats` và `standard_attributes`: Đại diện cho các thống kê tiêu chuẩn của cầu thủ, bao gồm thông tin cơ bản và hiệu suất thi đấu:
  - `player`: Tên cầu thủ.
  - `nationality`: Quốc tịch, được trích xuất từ thẻ `<span>` trong thẻ `<a>` bằng hàm lambda tùy chỉnh.
  - `team`: Tên đội bóng.
  - `position`: Vị trí thi đấu (ví dụ: FW, MF, GK).
  - `birth_year`: Năm sinh, tính bằng 2025 trừ đi tuổi từ thẻ `<td>` bằng hàm lambda.
  - `games`, `games_starts`, `minutes`: Số trận đấu, số trận đá chính, và số phút thi đấu.
  - `goals`, `assists`: Số bàn thắng và kiến tạo.
  - `cards_yellow`, `cards_red`: Số thẻ vàng và thẻ đỏ.
  - `xg`, `xg_assist`: Bàn thắng kỳ vọng (expected goals) và kiến tạo kỳ vọng.
  - `progressive_carries`, `progressive_passes`, `progressive_passes_received`: Số lần mang bóng, chuyền bóng, và nhận chuyền bóng tiến lên phía trước.

  - `goals_per90`, `assists_per90`, `xg_per90`, `xg_assist_per90`: Các chỉ số bàn thắng, kiến tạo, và kỳ vọng trên 90 phút.

  Vai trò: Cung cấp dữ liệu tổng quan về hiệu suất và thông tin cá nhân của cầu thủ, dùng làm cơ sở chính để hợp nhất với các bảng khác.

- `goalkeeping_attributes`: Thuộc tính cho thống kê của thủ môn:
  - `player`, `team`, `position`: Như trên, để xác định danh tính và đội.
  - `gk_goals_against_per90`: Số bàn thua trung bình mỗi 90 phút.
  - `gk_save_pct`: Tỷ lệ cứu thua.
  - `gk_clean_sheets_pct`: Tỷ lệ giữ sạch lưới.
  - `gk_pens_save_pct`: Tỷ lệ cứu penalty.

  Vai trò: Thu thập dữ liệu chuyên biệt cho thủ môn, dùng để phân tích hiệu suất phòng ngự của họ.

- `shooting_attributes`: Thuộc tính cho thống kê sút bóng:
  - `player`, `team`, `position`: Như trên.
  - `shots_on_target_pct`: Tỷ lệ sút trúng đích.
  - `shots_on_target_per90`: Số cú sút trúng đích mỗi 90 phút.
  - `goals_per_shot`: Tỷ lệ ghi bàn trên mỗi cú sút.
  - `average_shot_distance`: Khoảng cách trung bình của các cú sút.

  Vai trò: Đánh giá khả năng sút bóng và hiệu quả ghi bàn của cầu thủ.

- `passing_attributes`: Thuộc tính cho thống kê chuyền bóng:
  - `player`, `team`, `position`: Như trên.
  - `passes_completed`, `passes_pct`: Số đường chuyền hoàn thành và tỷ lệ chuyền chính xác.
  - `passes_total_distance`: Tổng khoảng cách của các đường chuyền.
  - `passes_pct_short`, `passes_pct_medium`, `passes_pct_long`: Tỷ lệ chính xác của chuyền ngắn, trung bình, dài.
  - `assisted_shots`: Số đường chuyền dẫn đến cú sút.
  - `passes_into_final_third`, `passes_into_penalty_area`: Số đường chuyền vào 1/3 cuối sân và khu vực penalty.
  - `crosses_into_penalty_area`: Số quả tạt vào khu vực penalty.
  - `progressive_passes`: Số đường chuyền tiến lên phía trước.

  Vai trò: Phân tích kỹ năng chuyền bóng và đóng góp vào tấn công của cầu thủ.

- `gca_attributes`: Thuộc tính cho thống kê tạo cơ hội ghi bàn:
  - `player`, `team`, `position`: Như trên.

- `sca`, `sca_per90`: Số hành động tạo cơ hội sút (shot-creating actions) và mỗi 90 phút.
- `gca`, `gca_per90`: Số hành động tạo cơ hội ghi bàn (goal-creating actions) và mỗi 90 phút.

Vai trò: Đánh giá khả năng tạo cơ hội dẫn đến bàn thắng hoặc cú sút.

- `defense_attributes`: Thuộc tính cho thống kê phòng ngự:
  - `player`, `team`, `position`: Như trên.
  - `tackles`, `tackles_won`: Số lần tắc bóng và số lần tắc bóng thành công.
  - `challenges`, `challenges_lost`: Số lần đối đầu và số lần thua đối đầu.
  - `blocks`, `blocked_shots`, `blocked_passes`: Số lần chặn bóng, chặn cú sút, chặn đường chuyền.
  - `interceptions`: Số lần đánh chặn.

  Vai trò: Đánh giá hiệu quả phòng ngự của cầu thủ.

- `possession_attributes`: Thuộc tính cho thống kê sở hữu bóng:
  - `player`, `team`, `position`: Như trên.
  - `touches`, `touches_def_pen_area`, `touches_def_3rd`, `touches_mid_3rd`, `touches_att_3rd`, `touches_att_pen_area`: Số lần chạm bóng, ở khu vực penalty phòng ngự, 1/3 phòng ngự, 1/3 giữa sân, 1/3 tấn công, và khu vực penalty tấn công.
  - `take_ons`, `take_ons_won`, `take_ons_tackled`: Số lần đi bóng, đi bóng thành công, và bị tắc bóng.
  - `carries`, `carries_progressive_distance`, `progressive_carries`: Số lần mang bóng, khoảng cách mang bóng tiến lên, và số lần mang bóng tiến lên.
  - `carries_into_final_third`, `carries_into_penalty_area`: Số lần mang bóng vào 1/3 cuối sân và khu vực penalty.
  - `miscontrols`, `dispossessed`: Số lần kiểm soát bóng hỏng và bị mất bóng.
  - `passes_received`, `progressive_passes_received`: Số đường chuyền nhận được và số đường chuyền tiến lên nhận được.

  Vai trò: Phân tích khả năng kiểm soát và di chuyển bóng của cầu thủ.

- `misc_attributes`: Thuộc tính cho thống kê khác:
  - `player`, `team`, `position`: Như trên.
  - `fouls`, `fouled`: Số lần phạm lỗi và bị phạm lỗi.
  - `offsides`: Số lần việt vị.
  - `crosses`: Số quả tạt.
  - `ball_recoveries`: Số lần thu hồi bóng.
  - `aerials_won`, `aerials_lost`, `aerials_won_pct`: Số lần thắng, thua tranh chấp trên không, và tỷ lệ thắng tranh chấp trên không.

Vai trò: Cung cấp các số liệu bổ sung về hành vi và hiệu suất của cầu thủ ngoài các hạng mục chính.

Các danh sách này được truyền vào hàm `create_attributes`, kết hợp với các hàm trích xuất tùy chỉnh (cho `nationality` và `birth_year`) hoặc mặc định, để tạo cặp (thuộc tính, hàm trích xuất). Chúng được sử dụng bởi hàm `scrape_table` để cào dữ liệu từ các bảng tương ứng, sau đó hợp nhất thành một tập dữ liệu tổng hợp về hiệu suất cầu thủ.

2.6 Phân chia các dataframe thành các trường dữ liệu từ các url và hợp nhất các dataframe

Code:

```
df_standard = scrape_table(driver,
    'https://fbref.com/en/comps/9/stats/Premier-League-Stats',
    standard_attributes)
df_goalkeeping = scrape_table(driver,
    'https://fbref.com/en/comps/9/keepers/Premier-League-Stats',
    goalkeeping_attributes, is_goalkeeper=True)
df_shooting = scrape_table(driver,
    'https://fbref.com/en/comps/9/shooting/Premier-League-Stats',
    shooting_attributes)
df_passing = scrape_table(driver,
    'https://fbref.com/en/comps/9/passing/Premier-League-Stats',
    passing_attributes)
df_gca = scrape_table(driver,
    'https://fbref.com/en/comps/9/gca/Premier-League-Stats',
    gca_attributes)
df_defense = scrape_table(driver,
    'https://fbref.com/en/comps/9/defense/Premier-League-Stats',
    defense_attributes)
df_possession = scrape_table(driver,
    'https://fbref.com/en/comps/9/possession/Premier-League-Stats',
    possession_attributes)
df_misc = scrape_table(driver,
    'https://fbref.com/en/comps/9/misc/Premier-League-Stats',
    misc_attributes)
```

Vai trò của các trường

Đoạn mã gọi hàm `scrape_table` nhiều lần để cào dữ liệu từ các bảng thống kê khác nhau trên trang fbref.com, mỗi bảng chứa một loại thông tin cụ thể về hiệu suất của cầu thủ trong giải Premier League. Các DataFrame được tạo ra từ các lời gọi này chứa dữ liệu tương ứng với các danh sách thuộc tính được định nghĩa trước đó, và vai trò của từng lời gọi như sau:

- `df_standard`: Thu thập dữ liệu từ bảng thống kê tiêu chuẩn tại `https://fbref.com/en/comps/9/stats/Premier-League-Stats`, sử dụng `standard_attributes`. Bảng này chứa các thông tin cơ bản và hiệu suất tổng quát của cầu thủ, bao gồm:

  - Tên cầu thủ, quốc tịch, đội bóng, vị trí, năm sinh.
  - Số trận đấu, trận đá chính, phút thi đấu.
  - Số bàn thắng, kiến tạo, thẻ vàng, thẻ đỏ.
  - Bàn thắng kỳ vọng (xG), kiến tạo kỳ vọng (xG assist).
  - Các chỉ số tiến bộ (mang bóng, chuyền bóng, nhận chuyền bóng tiến lên).
  - Các chỉ số mỗi 90 phút (bàn thắng, kiến tạo, xG, xG assist).

  Vai trò: Cung cấp tập dữ liệu nền tảng chứa thông tin chính của tất cả cầu thủ, làm cơ sở để hợp nhất với các bảng khác.

- `df_goalkeeping`: Thu thập dữ liệu từ bảng thống kê thủ môn tại `https://fbref.com/en/comps/9/keepers/Premier-League-Stats`, sử dụng `goalkeeping_attributes` và thiết lập `is_goalkeeper=True` để chỉ lấy dữ liệu của thủ môn (có vị trí `GK`). Bảng này chứa:

  - Tên cầu thủ, đội bóng, vị trí.
  - Số bàn thua mỗi 90 phút, tỷ lệ cứu thua, tỷ lệ giữ sạch lưới, tỷ lệ cứu penalty.

  Vai trò: Cung cấp dữ liệu chuyên biệt về hiệu suất của thủ môn, dùng để phân tích khả năng phòng ngự của họ.

- `df_shooting`: Thu thập dữ liệu từ bảng thống kê sút bóng tại `https://fbref.com/en/comps/9/shooting/Premier-League-Stats`, sử dụng `shooting_attributes`. Bảng này chứa:

  - Tên cầu thủ, đội bóng, vị trí.
  - Tỷ lệ sút trúng đích, số cú sút trúng đích mỗi 90 phút, tỷ lệ ghi bàn trên mỗi cú sút, khoảng cách trung bình của cú sút.

  Vai trò: Đánh giá kỹ năng sút bóng và hiệu quả ghi bàn của cầu thủ.

- `df_passing`: Thu thập dữ liệu từ bảng thống kê chuyền bóng tại `https://fbref.com/en/comps/9/passing/Premier-League-Stats`, sử dụng `passing_attributes`. Bảng này chứa:

  - Tên cầu thủ, đội bóng, vị trí.
  - Số đường chuyền hoàn thành, tỷ lệ chuyền chính xác, tổng khoảng cách chuyền.
  - Tỷ lệ chính xác của chuyền ngắn, trung bình, dài.
  - Số đường chuyền dẫn đến cú sút, chuyền vào 1/3 cuối sân, khu vực penalty, tạt vào penalty, và chuyền tiến lên.

  Vai trò: Phân tích khả năng chuyền bóng và đóng góp vào lối chơi tấn công của cầu thủ.

- `df_gca`: Thu thập dữ liệu từ bảng thống kê tạo cơ hội ghi bàn tại `https://fbref.com/en/comps/9/gca/Premier-League-Stats`, sử dụng `gca_attributes`. Bảng này chứa:

  - Tên cầu thủ, đội bóng, vị trí.
  - Số hành động tạo cơ hội sút (SCA) và ghi bàn (GCA), cùng chỉ số mỗi 90 phút.

Vai trò: Đánh giá khả năng tạo cơ hội dẫn đến cú sút hoặc bàn thắng.

- `df_defense`: Thu thập dữ liệu từ bảng thống kê phòng ngự tại `https://fbref.com/en/comps/9/defense/Premier-League-Stats`, sử dụng `defense_attributes`. Bảng này chứa:
  - Tên cầu thủ, đội bóng, vị trí.
  - Số lần tắc bóng, tắc bóng thành công, đối đầu, thua đối đầu.
  - Số lần chặn bóng, chặn cú sút, chặn đường chuyền, và đánh chặn.

  Vai trò: Đánh giá hiệu quả phòng ngự của cầu thủ.

- `df_possession`: Thu thập dữ liệu từ bảng thống kê sở hữu bóng tại `https://fbref.com/en/comps/9/possession/Premier-League-Stats`, sử dụng `possession_attributes`. Bảng này chứa:
  - Tên cầu thủ, đội bóng, vị trí.
  - Số lần chạm bóng ở các khu vực (phòng ngự, giữa sân, tấn công, penalty).
  - Số lần đi bóng, đi bóng thành công, bị tắc bóng.
  - Số lần mang bóng, khoảng cách mang bóng tiến lên, mang bóng vào 1/3 cuối sân, khu vực penalty.
  - Số lần kiểm soát bóng hỏng, bị mất bóng, nhận đường chuyền, và nhận chuyền tiến lên.

  Vai trò: Phân tích khả năng kiểm soát và di chuyển bóng của cầu thủ.

- `df_misc`: Thu thập dữ liệu từ bảng thống kê khác tại `https://fbref.com/en/comps/9/misc/Premier-League-Stats`, sử dụng `misc_attributes`. Bảng này chứa:
  - Tên cầu thủ, đội bóng, vị trí.
  - Số lần phạm lỗi, bị phạm lỗi, việt vị, tạt bóng.
  - Số lần thu hồi bóng, thắng/thua tranh chấp trên không, và tỷ lệ thắng tranh chấp trên không.

  Vai trò: Cung cấp các số liệu bổ sung về hành vi và hiệu suất của cầu thủ ngoài các hạng mục chính.

Mỗi lời gọi hàm `scrape_table` tạo ra một DataFrame chứa dữ liệu từ một bảng cụ thể, với các cột tương ứng với thuộc tính được định nghĩa trong danh sách `attributes` tương ứng. Các DataFrame này sau đó được hợp nhất (dựa trên các cột chung như `player`, `team`, `position`) để tạo thành một tập dữ liệu tổng hợp, cung cấp cái nhìn toàn diện về hiệu suất của các cầu thủ trong giải Premier League.

2.7 Xây dựng tiêu đề cho các trường dữ liệu

Code:

```
columns = [('', '', 'No.'), ('', '', 'Player')] + [
    ('', 'Nation', 'nationality'), ('', 'Team', 'team'), ('', 'Position', 'position'),
    ('', 'Age', 'birth_year'), ('', 'Playing Time', 'games'), ('', '', 'games_starts'), ('', '', 'minutes'),
    ('', 'Performance', 'goals'), ('', '', 'assists'), ('', '', 'cards_yellow'), ('', '', 'cards_red'),
    ('', 'Expected', 'xg'), ('', '', 'xg_assist'), ('', 'Progression', 'progressive_carries'),
    ('', '', 'progressive_passes'), ('', '', 'progressive_passes_received'), ('', 'Per 90 minutes', 'goals_per90'),
    ('', '', 'assists_per90'), ('', '', 'xg_per90'), ('', '', 'xg_assist_per90'),
    ('Goalkeeping', 'Performance', 'gk_goals_against_per90'), ('', '', 'gk_save_pct'), ('', '', 'gk_clean_sheets_pct'),
    ('', 'Penalty Kicks', 'gk_pens_save_pct'), ('Shooting', 'Standard', 'shots_on_target_pct'),
    ('', '', 'shots_on_target_per90'), ('', '', 'goals_per_shot'), ('', '', 'average_shot_distance'),
    ('Passing', 'Total', 'passes_completed'), ('', '', 'passes_pct'), ('', '', 'passes_total_distance'),
    ('', 'Short', 'passes_pct_short'), ('', 'Medium', 'passes_pct_medium'), ('', 'Long', 'passes_pct_long'),
    ('', 'Expected', 'assisted_shots'), ('', '', 'passes_into_final_third'), ('', '', 'passes_into_penalty_area'),
    ('', '', 'crosses_into_penalty_area'), ('', '', 'progressive_passes'), ('Goal and Shot Creation', 'SCA', 'sca'),
    ('', '', 'sca_per90'), ('', 'GCA', 'gca'), ('', '', 'gca_per90'), ('Defensive Actions', 'Tackles', 'tackles'),
    ('', '', 'tackles_won'), ('', 'Challenges', 'challenges'), ('', '', 'challenges_lost'),
    ('', 'Blocks', 'blocks'), ('', '', 'blocked_shots'), ('', '', 'blocked_passes'), ('', '', 'interceptions'),
    ('Possession', 'Touches', 'touches'), ('', '', 'touches_def_pen_area'), ('', '', 'touches_def_3rd'),
    ('', '', 'touches_mid_3rd'), ('', '', 'touches_att_3rd'), ('', '', 'touches_att_pen_area'),
    ('', 'Take-Ons', 'take_ons'), ('', '', 'take_ons_won'), ('', '', 'take_ons_tackled'),
    ('', 'Carries', 'carries'), ('', '', 'carries_progressive_distance'), ('', '', 'progressive_carries'),
    ('', '', 'carries_into_final_third'), ('', '', 'carries_into_penalty_area'), ('', '', 'miscontrols'),
    ('', '', 'dispossessed'), ('', 'Receiving', 'passes_received'), ('', '', 'progressive_passes_received'),
    ('Miscellaneous Stats', 'Performance', 'fouls'), ('', '', 'fouled'), ('', '', 'offsides'), ('', '', 'crosses'),
    ('', '', 'ball_recoveries'), ('', 'Aerial Duels', 'aerials_won'), ('', '', 'aerials_lost'),
    ('', '', 'aerials_won_pct')
]

```

```
multi_cols = pd.MultiIndex.from_tuples(columns)
tmp.columns = multi_cols

# Saving to CSV file
tmp.to_csv('results.csv', index=False)
```

3 Chương 2: Xử lý dữ liệu và thống kê dữ liệu

Chương này sẽ đi vào xử lý dữ liệu và thống kê các chỉ số theo yêu cầu: chọn ra 3 cầu thủ có chỉ số cao nhất và 3 cầu thủ có chỉ số thấp nhất cho mỗi cột dữ liệu. Tính toán giá trị trung bình, độ lệch chuẩn cho mỗi giá trị thống kê của tất cả cầu thủ cho mỗi đội, trực quan hoá dữ liệu bằng biểu đồ histogram với mỗi thuộc tính của tất cả cầu thủ và mỗi đội. Phần này chia ra làm 3 ý chính về hướng giải quyết

- Problem - 1: Thống kê ra 3 cầu thủ có chỉ số cao nhất và thấp nhất.
- Problem - 2: Tính toán giá trị trung bình, trung vị và độ lệch chuẩn.
- Problem - 3: Trực quan hoá dữ liệu theo từng chỉ số.

3.1 Tổng quan về code - Problem 1:

```
df_copy_01 = df.copy()

numeric_columns = [col for col in df_copy_01.columns if col not in
    ['player', 'nationality', 'team', 'position']]
df_copy_01[numeric_columns] =
    df_copy_01[numeric_columns].apply(pd.to_numeric,
    errors='coerce').fillna(0)

top_3_results = []
for col in numeric_columns:
    top_high = df_copy_01[['player', col]].sort_values(by=col,
        ascending=False).head(3)
    top_high = top_high.reset_index(drop=True)

    # top_low = df_copy_01[df_copy_01[col] > 0][['player',
        col]].sort_values(by=col, ascending=True).head(3)
    top_low = df_copy_01[['player', col]].sort_values(by=col,
        ascending=True).head(3)
    top_low = top_low.reset_index(drop=True)

    top_3_results.append(f"Statistic: {col}\n")
    top_3_results.append("Top 3 Highest:\n")
    for i, row in top_high.iterrows():
        top_3_results.append(f"{i+1}. {row['player']} ({row[col]})\n")
    top_3_results.append("Top 3 Lowest:\n")
    for i, row in top_low.iterrows():
        top_3_results.append(f"{i+1}. {row['player']} ({row[col]})\n")
    top_3_results.append("\n")
```

```

with open('top_3.txt', 'w', encoding='utf-8') as f:
    f.writelines(top_3_results)

driver.quit()
```

Vai trò của đoạn mã

Đoạn mã trên xử lý dataframe để tìm ra 3 cầu thủ có giá trị chỉ số cao nhất và thấp nhất cho mỗi thuộc tính. Để tránh thay đổi dữ liệu gốc, đoạn mã tạo bản sao bằng cách lưu vào df-copy-01 từ dataframe df để tránh thay đổi dữ liệu gốc. tiếp theo, mã xác định các cột số học bằng cách lọc ra tất cả cột trừ các cột không phải số như player, nationality, team, position. các cột này được chuyển đổi thành dạng số, với giá trị không hợp lệ thành nan và được thay bằng 0. sau đó, mã duyệt qua từng cột số học, sắp xếp để lấy ba cầu thủ có giá trị cao nhất và thấp nhất, định dạng kết quả thành chuỗi văn bản với tên cầu thủ và giá trị tương ứng, ví dụ: "1. messi (15)". các chuỗi này được lưu vào danh sách top3.txt bao gồm tiêu đề cho mỗi chỉ số, danh sách ba cầu thủ cao nhất và thấp nhất, cùng các dòng trống để phân tách. cuối cùng, danh sách được ghi vào file top-3.txt với mã hóa utf-8 để hỗ trợ ký tự đặc biệt, và trình duyệt selenium được đóng để giải phóng tài nguyên. đoạn mã giúp cung cấp thông tin về những cầu thủ nổi bật và kém nổi bật theo từng chỉ số, hỗ trợ đánh giá hiệu suất trong giải premier league.

3.2 Tổng quan về code - Problem 2:

```
import pandas as pd
import numpy as np

# Create a copy of the original dataframe
df_copy = df.copy()

# Process the copied dataframe
numeric_columns = [col for col in df_copy.columns if col not in
    ['player', 'nationality', 'team', 'position']]
df_copy[numeric_columns] =
    df_copy[numeric_columns].apply(pd.to_numeric,
    errors='coerce').fillna(0)

teams = df_copy['team'].unique()
stats_columns = numeric_columns

results_data = []

all_stats = {}
for stat in stats_columns:
    all_stats[('Median', stat)] = df_copy[stat].median()
    all_stats[('Mean', stat)] = df_copy[stat].mean()
    all_stats[('Std', stat)] = df_copy[stat].std()
results_data.append(['all'] + [all_stats.get((metric, stat), 0) for
    stat in stats_columns for metric in ['Median', 'Mean', 'Std']])

```

```
for team in teams:
    team_df = df_copy[df_copy['team'] == team]
    team_stats = {}
    for stat in stats_columns:
        team_stats[('Median', stat)] = team_df[stat].median()
        team_stats[('Mean', stat)] = team_df[stat].mean()
        team_stats[('Std', stat)] = team_df[stat].std()
    results_data.append([team] + [team_stats.get((metric, stat), 0) for
       stat in stats_columns for metric in ['Median', 'Mean', 'Std']])

columns = pd.MultiIndex.from_tuples(
    [('', '')] + [(metric, stat) for stat in stats_columns for metric
       in ['Median', 'Mean', 'Std']]
)

results_df = pd.DataFrame(results_data, columns=columns)
results_df.iloc[:, 0] = ['all'] + list(teams)

results_df.to_csv('results2.csv', index=False)
```

Đoạn mã này xử lý DataFrame chứa dữ liệu thống kê bóng đá từ giải Premier League để tính toán các giá trị thống kê mô tả, bao gồm trung vị, trung bình, và độ lệch chuẩn cho các chỉ số số học của tất cả cầu thủ và theo từng đội. Kết quả được tổ chức vào một DataFrame mới với cấu trúc cột phân cấp (MultiIndex) và lưu vào file **results2.csv**. Mục tiêu là cung cấp thông tin tổng quan về phân bố các chỉ số, hỗ trợ phân tích hiệu suất cầu thủ và so sánh giữa các đội.

Vai trò của đoạn mã

Đoạn mã thực hiện các bước để tính toán và tổ chức các giá trị thống kê mô tả (trung vị, trung bình, độ lệch chuẩn) cho các chỉ số số học trong DataFrame **df**, cả ở cấp độ toàn giải (tất cả cầu thủ) và cấp độ đội. Dữ liệu được lưu vào file **results2.csv** với cấu trúc cột phân cấp, giúp dễ dàng phân tích và so sánh hiệu suất cầu thủ giữa các đội. Các bước chi tiết bao gồm:

- **Tạo bản sao DataFrame**: Tạo bản sao **df-copy** từ **df** bằng **df.copy()** để tránh thay đổi dữ liệu gốc, đảm bảo an toàn cho các phân tích khác.
- **Xác định và xử lý cột số học**: Sử dụng list comprehension để tạo danh sách **numeric-columns**, bao gồm tất cả cột trừ các cột văn bản (**player**, **nationality**, **team**, **position**). Các cột này được chuyển thành dạng số bằng **pd.to-numeric** với **errors='coerce'**, thay các giá trị không hợp lệ thành **NaN**, sau đó điền **NaN** bằng 0 bằng **fillna(0)**. Điều này đảm bảo dữ liệu sẵn sàng cho tính toán thống kê.
- **Xác định danh sách đội**: Tạo danh sách **teams** chứa các giá trị duy nhất trong cột **team** bằng **df-copy['team'].unique()**. Gán **stats-columns** bằng **numeric-columns** để sử dụng trong các bước tính toán.
- **Tính thống kê cho tất cả cầu thủ**: Tạo từ điển **all-stats** để lưu trữ trung vị (**median**), trung bình (**mean**), và độ lệch chuẩn (**std**) cho mỗi chỉ số trong **stats-columns**. Các giá trị được tính trên toàn bộ **df-copy** và lưu vào danh sách

results-data dưới dạng một hàng với nhãn 'all', theo thứ tự trung vị, trung bình, độ lệch chuẩn cho từng chỉ số.

- **Tính thống kê theo đội**: Duyệt qua từng đội trong **teams**, lọc dữ liệu của đội đó vào **team-df** bằng **df-copy**[**df-copy**['**team**'] == **team**]. Tạo từ điển **team-stats** để tính trung vị, trung bình, và độ lệch chuẩn cho mỗi chỉ số. Kết quả được thêm vào **results-data** với nhãn là tên đội, theo định dạng tương tự hàng 'all'.
- **Tạo cấu trúc cột MultiIndex**: Tạo danh sách **columns** với cấu trúc MultiIndex, bắt đầu bằng cột rỗng cho tên đội hoặc 'all', tiếp theo là các cặp (**metric**, **stat**) cho mỗi chỉ số và mỗi loại thống kê (**Median**, **Mean**, **Std**). MultiIndex được tạo bằng **pd.MultiIndex.from-tuples**.
- **Tạo DataFrame kết quả**: Tạo **results-df** từ **results-data** với cột là **columns**. Gán cột đầu tiên bằng danh sách ['**all**'] + **list(teams)** để hiển thị nhãn 'all' và tên các đội. Lưu DataFrame vào file **results2.csv** bằng **to-csv** mà không bao gồm chỉ số.

Đoạn mã cung cấp một cách tiếp cận có hệ thống để tổng hợp và trình bày các thống kê mô tả, giúp phân tích sự phân bố của các chỉ số hiệu suất cầu thủ ở cả cấp độ toàn giải và cấp độ đội, hỗ trợ việc so sánh và đánh giá.

Thống kê các Team có chỉ số cao nhất theo từng chỉ số thống kê

```python
df = df_standard
for df_temp in [df_goalkeeping, df_shooting, df_passing, df_gca, df_defense, df_possession, df_misc]:
    df = df.merge(df_temp, on=['player', 'team', 'position'], how='left')

df = df.sort_values(by='player')

df_copy = df.copy()
numeric_columns = [col for col in df_copy.columns if col not in ['player', 'nationality', 'team', 'position']]
df_copy[numeric_columns] = df_copy[numeric_columns].apply(pd.to_numeric, errors='coerce').fillna(0)

results = []
for stat in numeric_columns:
    team_stats = df_copy.groupby('team')[stat].sum().reset_index()
    max_value = team_stats[stat].max()
    top_teams = team_stats[team_stats[stat] == max_value]
    results.append(f"Statistic: {stat}\n")
    for _, row in top_teams.iterrows():
        results.append(f"Top Team: {row['team']} ({row[stat]})\n")
    results.append("\n")

with open('highest_score_team.txt', 'w', encoding='utf-8') as f:
    f.writelines(results)
```

1. **Gộp dữ liệu**:
   - Dòng `df = df_standard` gán DataFrame `df_standard` vào `df`.
   - Vòng lặp `for df_temp in [...]` gộp `df` với từng DataFrame như `df_goalkeeping`, `df_shooting`, ... bằng phương thức `merge` trên các cột `player`, `team`, `position`. Tham số `how='left'` giữ tất cả hàng từ `df` và bổ sung dữ liệu từ `df_temp`.
2. **Sắp xếp dữ liệu**:
   - `df = df.sort_values(by='player')` sắp xếp DataFrame theo cột `player` theo thứ tự tăng dần.
3. **Xử lý dữ liệu số**:
   - `df_copy = df.copy()` tạo bản sao của `df`.
   - `numeric_columns = [col for col in df_copy.columns if col not in [...]]` lấy các cột không phải `player`, `nationality`, `team`, `position`.
   - `df_copy[numeric_columns].apply(pd.to_numeric, errors='coerce').fillna(0)` chuyển các giá trị trong `numeric_columns` thành số, giá trị không hợp lệ thành `NaN`, sau đó điền `NaN` bằng 0.
4. **Tính toán và lưu kết quả**:
   - Vòng lặp `for stat in numeric_columns` duyệt qua từng cột số.
   - `team_stats = df_copy.groupby('team')[stat].sum().reset_index()` tính tổng `stat` cho mỗi đội.
   - `max_value = team_stats[stat].max()` tìm giá trị lớn nhất của `stat`.
   - `top_teams = team_stats[team_stats[stat] == max_value]` lọc các đội có giá trị `stat` tối đa.
   - Kết quả được lưu vào danh sách `results` và ghi vào file `highest_score_team.txt`.

Phân tích đội bóng xuất sắc nhất Premier League 2024-2025

Dựa trên phân tích, đội bóng đang thể hiện tốt nhất trong mùa giải **Premier League 2024-2025** là **Liverpool FC**. Liverpool đã giành chức vô địch sớm với 4 trận còn lại, đạt **82 điểm** sau 34 trận, với hiệu số bàn thắng bại **+48**. Họ thể hiện sự ổn định vượt trội, dẫn đầu bảng xếp hạng và có hàng công mạnh mẽ với **80 bàn thắng**, cùng hàng thủ chắc chắn chỉ để thủng lưới **32 bàn**. Dưới sự dẫn dắt của huấn luyện viên **Arne Slot**, Liverpool đã vượt qua những thách thức lớn, bao gồm việc thay thế **Jürgen Klopp**, và duy trì phong độ ấn tượng cả trong nước lẫn châu Âu.

3.3 Tổng quan về code - Problem 3:

3.3.1 Trực quan hoá dữ liệu thống kê từ tất cả các cầu thủ

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def plot_team_histograms(df):
    attacking_stats = ['goals', 'assists', 'xg']
    defensive_stats = ['tackles', 'interceptions', 'blocks']
    all_stats = attacking_stats + defensive_stats
    teams = df['team'].unique()

    for stat in all_stats:
        fig, axes = plt.subplots(4, 5, figsize=(25, 20))
        fig.patch.set_facecolor('#f5f5f5')
        fig.suptitle(f'Histograms for {stat.upper()} by Team',
            fontsize=16, y=1.02)
        axes = axes.flatten()

        color = 'red' if stat in attacking_stats else 'navy'

        for i, team in enumerate(teams):
            team_data = df[df['team'] == team][stat]
            axes[i].hist(team_data, bins=20, alpha=0.7, color=color,
                density=True, edgecolor='black')
            axes[i].set_title(f'{team}', fontsize=12, pad=10)
            axes[i].set_xlabel(stat, fontsize=10)
            axes[i].set_ylabel('Density', fontsize=10)
            axes[i].grid(True, alpha=0.3)
            axes[i].set_facecolor('#ffffff')

        for i in range(len(teams), 20):
            axes[i].set_visible(False)

        plt.tight_layout()
        plt.show()

df_copy_03 = df_copy_01.copy()
plot_team_histograms(df_copy_03)
```

Đoạn mã này sử dụng thư viện matplotlib để tạo biểu đồ histogram, trực quan hóa phân bố của các chỉ số thống kê bóng đá từ DataFrame trong giải Premier League. Nó tập trung vào sáu chỉ số: ba chỉ số tấn công (goals, assists, xg) và ba chỉ số phòng ngự (tackles, interceptions, blocks). Mã tạo một lưới 2x3 biểu đồ, mỗi biểu đồ hiển thị histogram của một chỉ số, với màu sắc khác nhau cho nhóm tấn công (đỏ) và phòng ngự (xanh navy). Các histogram được chuẩn hóa theo mật độ, có lưới nền và các chi tiết tùy chỉnh để tăng tính trực quan. Mục tiêu là cung cấp cái nhìn trực quan về phân bố của các chỉ số, giúp phân tích sự khác biệt trong hiệu suất cầu thủ.

Vai trò của đoạn mã

Đoạn mã thực hiện việc trực quan hóa phân bố của các chỉ số tấn công và phòng ngự trong DataFrame **df-copy-02** bằng biểu đồ histogram, giúp phân tích sự khác biệt trong

hiệu suất cầu thủ Premier League. Các bước chi tiết bao gồm:

- **Tạo bản sao DataFrame**: Tạo bản sao **df-copy-02** từ **df-copy-01** bằng **df.copy()** để bảo vệ dữ liệu gốc, đảm bảo các thay đổi không ảnh hưởng đến DataFrame ban đầu.
- **Định nghĩa hàm plot-stat-histograms**: Hàm này nhận DataFrame làm đầu vào và thực hiện các bước:
  - Xác định hai danh sách chỉ số: **attacking-stats** (goals, assists, xg) và **defensive-stats** (tackles, interceptions, blocks).
  - Tạo lưới biểu đồ 2x3 bằng **plt.subplots** với kích thước 18x8 inch, đặt nền tổng thể màu xám nhạt (**#f5f5f5**).
  - Với mỗi chỉ số trong **attacking-stats**, vẽ histogram trên hàng đầu tiên (axes[0, i]) với 20 bin, màu đỏ, độ trong suốt 0.7, chuẩn hóa theo mật độ (**density=True**), và viền đen. Thiết lập tiêu đề (**Distribution of {stat}**), nhãn trục x (**stat**), nhãn trục y (**Density**), lưới nền mờ (**alpha=0.3**), và nền biểu đồ màu trắng (**#ffffff**).
  - Với mỗi chỉ số trong **defensive-stats**, vẽ histogram trên hàng thứ hai (axes[1, i]) tương tự, nhưng dùng màu xanh navy.
  - Sử dụng **plt.tight-layout** để tối ưu hóa khoảng cách giữa các biểu đồ, tránh chồng lấn.
  - Hiển thị hình bằng **plt.show**.

  Vai trò: Tạo các biểu đồ histogram trực quan, phân biệt rõ ràng giữa chỉ số tấn công và phòng ngự, giúp nhận diện xu hướng và phân bố của dữ liệu.
- **Thực thi hàm**: Gọi **plot-stat-histograms** với **df-copy-02** để tạo và hiển thị lưới 2x3 biểu đồ histogram, trong đó mỗi biểu đồ cho thấy phân bố của một chỉ số cụ thể. Vai trò: Tạo ra sản phẩm trực quan cuối cùng, cung cấp cái nhìn rõ ràng về dữ liệu cho người dùng.

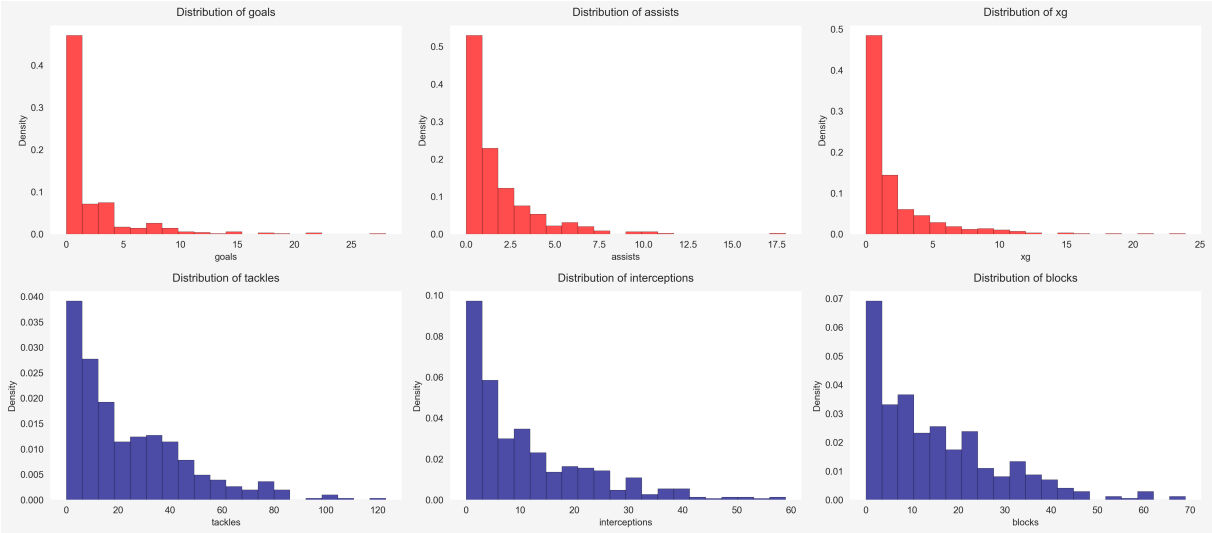
Đoạn mã này giúp trực quan hóa dữ liệu một cách hiệu quả, sử dụng màu sắc và bố cục rõ ràng để phân biệt các nhóm chỉ số, hỗ trợ phân tích và trình bày kết quả về hiệu suất cầu thủ trong giải Premier League.

Hình ảnh sau khi trực quan hoá dữ liệu

3.3.2 Trực quan hoá dữ liệu cho từng chỉ số thống kê của 20 câu lạc bộ

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def plot_team_histograms(df):
    attacking_stats = ['goals', 'assists', 'xg']
    defensive_stats = ['tackles', 'interceptions', 'blocks']
    all_stats = attacking_stats + defensive_stats
```



Hình 1: Thống kê chỉ số của tất cả cầu thủ của từng chỉ số

```
    teams = df['team'].unique()

    for stat in all_stats:
        fig, axes = plt.subplots(4, 5, figsize=(25, 20))
        fig.patch.set_facecolor('#f5f5f5')
        fig.suptitle(f'Histograms for {stat.upper()} by Team',
            fontsize=16, y=1.02)
        axes = axes.flatten()

        color = 'red' if stat in attacking_stats else 'navy'

        for i, team in enumerate(teams):
            team_data = df[df['team'] == team][stat]
            axes[i].hist(team_data, bins=20, alpha=0.7, color=color,
                density=True, edgecolor='black')
            axes[i].set_title(f'{team}', fontsize=12, pad=10)
            axes[i].set_xlabel(stat, fontsize=10)
            axes[i].set_ylabel('Density', fontsize=10)
            axes[i].grid(True, alpha=0.3)
            axes[i].set_facecolor('#ffffff')

        for i in range(len(teams), 20):
            axes[i].set_visible(False)

        plt.tight_layout()
        plt.show()

df_copy_03 = df_copy_01.copy()
plot_team_histograms(df_copy_03)
```

Đoạn mã này sử dụng thư viện matplotlib để tạo biểu đồ histogram, trực quan hóa phân bố của các chỉ số thống kê bóng đá (goals, assists, xg, tackles, interceptions, blocks) cho từng đội trong giải Premier League. Mã tạo một lưới 4x5 biểu đồ cho mỗi chỉ số, với mỗi biểu đồ hiển thị histogram của một đội, sử dụng màu đỏ cho chỉ số tấn công và xanh

navy cho chỉ số phòng ngự. Các histogram được chuẩn hóa theo mật độ, có tiêu đề, nhãn trục, lưới nền, và các tùy chỉnh giao diện để tăng tính trực quan. Mục tiêu là cung cấp cái nhìn trực quan về phân bố các chỉ số theo đội, giúp so sánh hiệu suất giữa các đội.

Vai trò của đoạn mã

Đoạn mã thực hiện việc trực quan hóa phân bố của các chỉ số thống kê bóng đá trong DataFrame **df-copy-03** bằng biểu đồ histogram, với mỗi biểu đồ đại diện cho một đội trong giải Premier League. Nó giúp so sánh hiệu suất của các đội qua các chỉ số tấn công và phòng ngự. Các bước chi tiết bao gồm:

- **Tạo bản sao DataFrame**: Tạo bản sao **df-copy-03** từ **df-copy-01** bằng **df.copy()** để bảo vệ dữ liệu gốc, đảm bảo các thay đổi không ảnh hưởng đến DataFrame ban đầu.
- **Định nghĩa hàm plot-team-histograms**: Hàm này nhận DataFrame làm đầu vào và thực hiện các bước:
  - Xác định danh sách chỉ số: **attacking-stats** (goals, assists, xg), **defensive-stats** (tackles, interceptions, blocks), và **all-stats** bằng cách nối hai danh sách.
  - Lấy danh sách đội duy nhất từ cột **team** bằng **df['team'].unique()**.
  - Duyệt qua từng chỉ số trong **all-stats**:
    * Tạo lưới biểu đồ 4x5 (20 ô) bằng **plt.subplots** với kích thước 25x20 inch, đặt nền tổng thể màu xám nhạt (**#f5f5f5**).
    * Đặt tiêu đề chung cho hình bằng **fig.suptitle** với nội dung **Histograms for {stat.upper()} by Team**.
    * Chuyển mảng axes thành dạng 1 chiều bằng **axes.flatten()** để dễ truy cập.
    * Chọn màu sắc: đỏ cho chỉ số trong **attacking-stats**, xanh navy cho **defensive-stats**.
    * Duyệt qua từng đội trong **teams**:
      · Lọc dữ liệu của đội bằng **df[df['team'] == team][stat]**.
      · Vẽ histogram trên ô **axes[i]** với 20 bin, độ trong suốt 0.7, màu theo chỉ số, chuẩn hóa theo mật độ (**density=True**), và viền đen.
      · Đặt tiêu đề là tên đội, nhãn trục x là tên chỉ số, nhãn trục y là **Density**, thêm lưới nền mờ (**alpha=0.3**), và nền biểu đồ màu trắng (**#ffffff**).
    * Ẩn các ô không sử dụng (nếu số đội dưới 20) bằng **axes[i].set-visible(False)**.
    * Sử dụng **plt.tight-layout** để tối ưu hóa bố cục, tránh chồng lấn.
    * Hiển thị hình bằng **plt.show**.

  Vai trò: Tạo bộ biểu đồ histogram riêng cho mỗi chỉ số, với mỗi biểu đồ hiển thị phân bố của một đội, giúp dễ dàng so sánh giữa các đội.
- **Thực thi hàm**: Gọi **plot-team-histograms** với **df-copy-03** để tạo và hiển thị một bộ biểu đồ 4x5 cho mỗi chỉ số trong **all-stats**, trong đó mỗi ô cho thấy phân bố của chỉ số đó cho một đội. Vai trò: Tạo ra sản phẩm trực quan cuối cùng, cung cấp cái nhìn chi tiết về hiệu suất của từng đội qua các chỉ số.

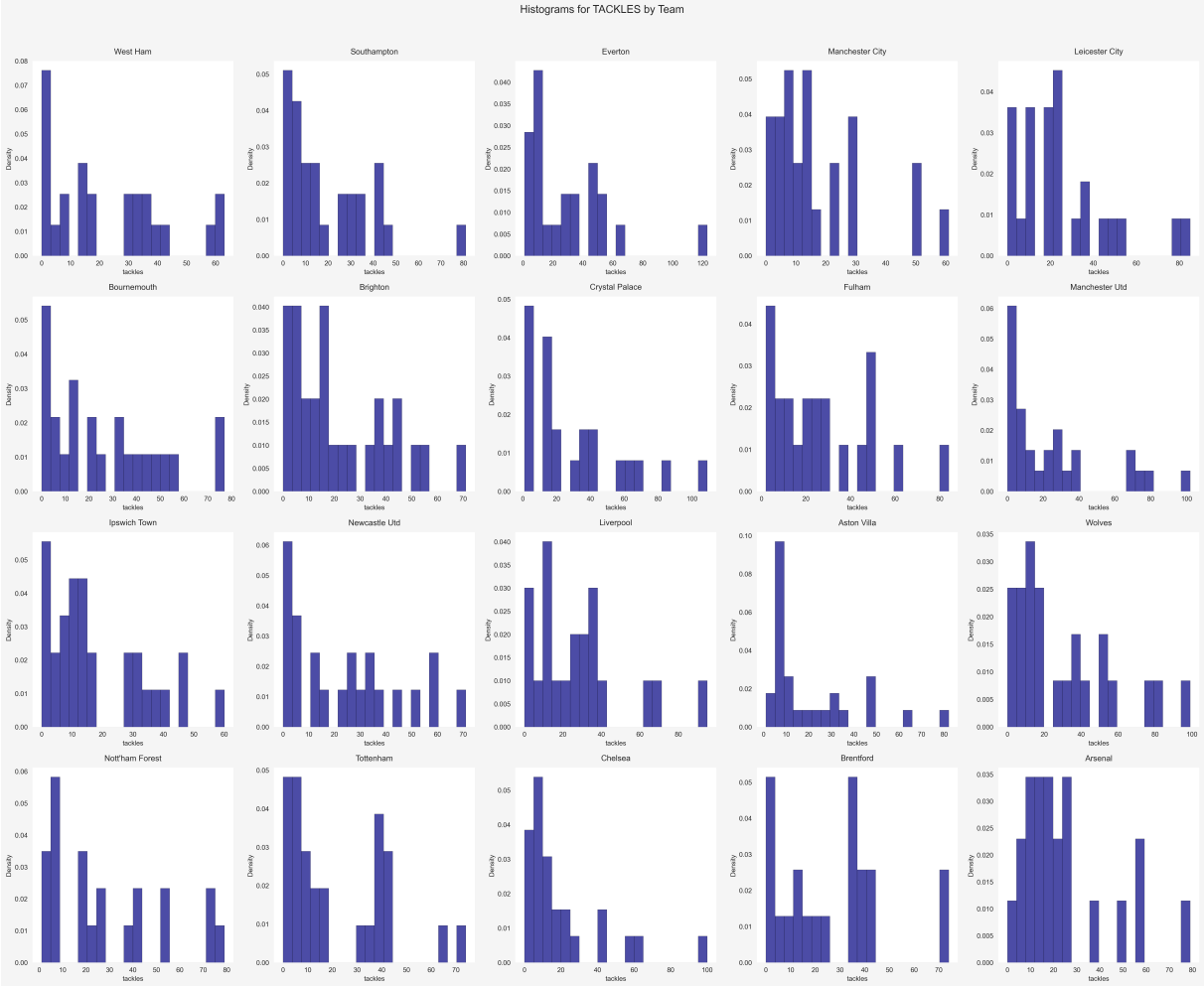
Đoạn mã này cung cấp một cách trực quan hóa hiệu quả, sử dụng lưới biểu đồ để hiển thị phân bố của từng chỉ số theo đội, với màu sắc phân biệt rõ ràng giữa chỉ số tấn công và phòng ngự, hỗ trợ phân tích và trình bày kết quả về hiệu suất trong giải Premier League.

Hình ảnh sau khi trực quan hoá dữ liệu

Trực quan hoá theo chỉ số: Goals

Hình 2: Thống kê chỉ số của tất cả các đội theo Goals

Trực quan hóa theo chỉ số: Tackles



Hình 3: Phân bố số lần tắc bóng của cầu thủ theo từng đội trong giải Premier League.

4 Chương 3: Phân cụm dữ liệu

Chương này sẽ đi vào xây dựng mô hình và thuật toán để phân cụm các cầu thủ (phân loại), sử dụng thuật toán K-means để phân loại các cầu thủ và PCA để giảm chiều dữ liệu rồi trực quan hoá dữ liệu.

4.1 Tổng quan về Code

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

df_03 = df.copy()

features = [
    'goals_per90', 'assists_per90', 'xg_per90', 'xg_assist_per90',
    'progressive_carries_x', 'progressive_passes_x',
        'progressive_passes_received_x',
    'shots_on_target_per90', 'goals_per_shot',
    'passes_completed', 'passes_pct', 'passes_into_final_third',
        'passes_into_penalty_area',
    'tackles', 'tackles_won', 'interceptions', 'blocks',
    'touches_att_3rd', 'touches_att_pen_area',
        'carries_into_penalty_area',
    'aerials_won', 'aerials_won_pct'
]

features = [f for f in features if f in df_03.columns]

for col in features:
    df_03[col] = pd.to_numeric(df_03[col], errors='coerce')
df_03[features] = df_03[features].fillna(0)

X = df_03[features].astype(float)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

inertia = []
K = range(1, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

optimal_k = 4
kmeans = KMeans(n_clusters=k, random_state=42)
df_03['Cluster'] = kmeans.fit_predict(X_scaled)
```

```python

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

sns.set_style("whitegrid")
plt.rcParams['axes.facecolor'] = '#f5f5f5'
plt.rcParams['figure.facecolor'] = '#ffffff'

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6), constrained_layout=True)

ax1.plot(K, inertia, 'o-', color='#1f77b4', markersize=10, linewidth=2.5, markeredgecolor='white')
ax1.set_xlabel('Clusters (k)', fontsize=12, weight='bold')
ax1.set_ylabel('Inertia', fontsize=12, weight='bold')
ax1.set_title('Elbow Method', fontsize=14, weight='bold', pad=10)
ax1.grid(True, linestyle='--', alpha=0.5)

scatter = ax2.scatter(X_pca[:, 0], X_pca[:, 1], c=df_03['Cluster'], cmap='Set2', s=80, alpha=0.9, edgecolors='white', linewidth=0.5)
cbar = fig.colorbar(scatter, ax=ax2)
cbar.set_label('Clusters', fontsize=12, weight='bold')
ax2.set_xlabel('Component 1', fontsize=12, weight='bold')
ax2.set_ylabel('Component 2', fontsize=12, weight='bold')
ax2.set_title('Players Clustering K-means (PCA 2D)', fontsize=14, weight='bold', pad=10)
ax2.grid(True, linestyle='--', alpha=0.5)

plt.suptitle('PLayers Clustering', fontsize=16, weight='bold', y=1.05)
plt.show()

result = df_03[['player', 'team', 'position', 'Cluster'] + features]

for cluster in range(optimal_k):
    cluster_data = result[result['Cluster'] == cluster]

result.to_csv('clustered_players.csv', index=False)
```

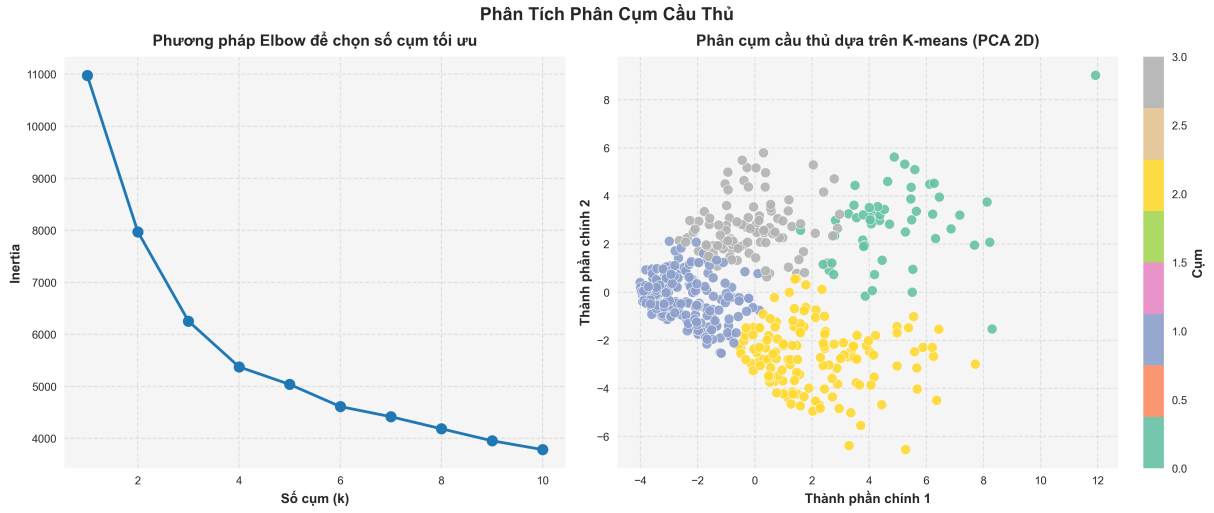
Đoạn mã này sử dụng thuật toán K-means để phân cụm cầu thủ trong giải Premier League dựa trên các chỉ số thống kê, trực quan hóa kết quả bằng biểu đồ Elbow và biểu đồ phân tán PCA 2D, sau đó lưu dữ liệu phân cụm vào file CSV. Mã xử lý DataFrame để chọn các đặc trưng, chuẩn hóa dữ liệu, xác định số cụm tối ưu, phân cụm, và hiển thị kết quả trực quan. Mục tiêu là nhóm các cầu thủ có hiệu suất tương tự vào các cụm, giúp phân tích vai trò và phong cách thi đấu của họ.

Vai trò của đoạn mã

Đoạn mã thực hiện phân cụm cầu thủ trong DataFrame **df-03** dựa trên các chỉ số thống kê, sử dụng thuật toán K-means, trực quan hóa kết quả bằng biểu đồ Elbow và biểu đồ phân tán PCA 2D, và lưu dữ liệu phân cụm vào file **clustered-players.csv**. Nó giúp nhóm các cầu thủ có phong cách thi đấu tương tự, hỗ trợ phân tích vai trò và hiệu suất. Các bước chi tiết bao gồm:

- **Tạo bản sao DataFrame**: Tạo bản sao **df-03** từ **df** bằng **df.copy()** để bảo vệ dữ liệu gốc, đảm bảo không thay đổi DataFrame ban đầu.
- **Xác định đặc trưng**: Định nghĩa danh sách **features** với 22 chỉ số thống kê liên quan đến tấn công (như **goals-per90**, **assists-per90**), chuyền bóng (**passes-completed**, **passes-pct**), phòng ngự (**tackles**, **interceptions**), và sở hữu bóng (**touches-att-3rd**, **carries-into-penalty-area**). Lọc các đặc trưng có trong **df-03** để tránh lỗi do cột không tồn tại.
- **Xử lý dữ liệu**: Chuyển các cột trong **features** thành dạng số bằng **pd.to-numeric** với **errors='coerce'**, thay giá trị không hợp lệ bằng **NaN**, sau đó điền **NaN** bằng 0 bằng **fillna(0)**. Tạo ma trận **X** từ **features**, đảm bảo tất cả giá trị là số thực (**astype(float)**).
- **Chuẩn hóa dữ liệu**: Sử dụng **StandardScaler** để chuẩn hóa **X** thành **X-scaled**, đưa các đặc trưng về cùng thang đo (trung bình 0, độ lệch chuẩn 1), đảm bảo K-means hoạt động hiệu quả.
- **Xác định số cụm tối ưu**: Chạy K-means với số cụm từ 1 đến 10, tính giá trị **inertia** (tổng bình phương khoảng cách từ điểm đến tâm cụm) cho mỗi **k**. Lưu kết quả vào danh sách **inertia**. Chọn **optimal-k = 4** làm số cụm tối ưu (dựa trên giả định từ phương pháp Elbow, không tự động chọn từ dữ liệu).
- **Phân cụm cầu thủ**: Chạy K-means với **n-clusters=4** và **random-state=42** trên **X-scaled**, gán nhãn cụm vào cột **Cluster** của **df-03** bằng **fit-predict**.
- **Giảm chiều bằng PCA**: Sử dụng **PCA** với **n-components=2** để giảm chiều **X-scaled** thành **X-pca**, giúp trực quan hóa dữ liệu trong không gian 2D.
- **Trực quan hóa kết quả**:
  - Tạo hình với hai biểu đồ bằng **plt.subplots** (kích thước 15x6 inch):
    - Biểu đồ trái (**ax1**): Vẽ đường cong Elbow bằng **plot**, hiển thị **inertia** theo **k** (1 đến 10), với điểm đánh dấu tròn, màu xanh dương, lưới nền mờ, tiêu đề **Phương pháp Elbow để chọn số cụm tối ưu**, và nhãn trục **Số cụm (k)** và **Inertia**.
    - Biểu đồ phải (**ax2**): Vẽ biểu đồ phân tán của **X-pca** với màu theo **Cluster**, sử dụng bảng màu **Set2**, kích thước điểm 80, độ trong suốt 0.9, viền trắng. Thêm thanh màu (**colorbar**) với nhãn **Cụm**, tiêu đề **Phân cụm cầu thủ dựa trên K-means (PCA 2D)**, và nhãn trục **Thành phần chính 1**, **Thành phần chính 2**.
  - Tùy chỉnh giao diện: Đặt nền biểu đồ màu xám nhạt (**#f5f5f5**), nền hình màu trắng (**#ffffff**), lưới nền mờ, sử dụng **constrained-layout** để tối ưu hóa bố cục, và thêm tiêu đề chung **Phân Tích Phân Cụm Cầu Thủ**.
  - Hiển thị hình bằng **plt.show**.
- **Xử lý và lưu kết quả**: Tạo DataFrame **result** chứa các cột **player**, **team**, **position**, **Cluster**, và các đặc trưng trong **features**. Lặp qua các cụm để lọc dữ liệu từng cụm (dù không sử dụng trong mã này). Lưu **result** vào file **clustered-players.csv** bằng **to-csv** mà không bao gồm chỉ số.

Đoạn mã này cung cấp một quy trình phân cụm hoàn chỉnh, từ xử lý dữ liệu, phân cụm, đến trực quan hóa và lưu kết quả, giúp phân tích và hiểu rõ hơn về các nhóm cầu thủ dựa trên hiệu suất trong giải Premier League.



5 Chương 4: Thu thập dữ liệu và xây dựng mô hình để dự đoán giá trị chuyển nhượng

Thu thập dữ liệu về giá trị chuyển nhượng của các cầu thủ trong mùa giải 2024 - 2025 và lấy ra các cầu thủ chơi bóng trên 900 phút. Xây dựng model để dự đoán giá trị chuyển nhượng.

5.1 Thu thập dữ liệu liên quan đến giá trị chuyển nhượng:

5.1.1 Tổng quan về Code:

```python
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup
import pandas as pd
import time

def setup_driver():
    chrome_options = Options()
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--disable-gpu')
    chrome_options.add_argument('--no-sandbox')
    driver = webdriver.Chrome(options=chrome_options)
    return driver

def scrape_page(driver, url):
    driver.get(url)
    time.sleep(3)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    table = soup.find('table', class_='table table-hover no-cursor table-striped leaguetable mvp-table similar-players-table mb-0')
    if not table:
        return []

    data = []
    rows = table.find('tbody').find_all('tr')

    for row in rows:
        try:
            skill_div = row.find('div', class_='table-skill__skill')
            pot_div = row.find('div', class_='table-skill__pot')
            skill = float(skill_div.text.strip()) if skill_div else None
            pot = float(pot_div.text.strip()) if pot_div else None
            skill_pot = f"{skill}/{pot}" if skill and pot else None

            player_span = row.find('span', class_='d-none')
            player_name = player_span.text.strip() if player_span else None

            team_span = row.find('span', class_='td-team__teamname')
            team = team_span.text.strip() if team_span else None
```

```python

            etv_span = row.find('span', class_='player-tag')
            etv = etv_span.text.strip() if etv_span else None

            data.append({
                'player_name': player_name,
                'team': team,
                'price': etv,
                'skill/pot': skill_pot
            })
        except:
            continue

    return data

base_url = "https://www.footballtransfers.com/en/players/uk-premier-league"
total_pages = 22
all_data = []

driver = setup_driver()

try:
    for page in range(1, total_pages + 1):
        if page == 1:
            url = base_url
        else:
            url = f"{base_url}/{page}"
        page_data = scrape_page(driver, url)
        all_data.extend(page_data)
finally:
    driver.quit()

df_final = pd.DataFrame(all_data)
df_final.to_csv('football_transfers_players.csv', index=False)
```

Listing 1: Web Scraping Football Transfers Players

5.1.2 Giải thích về Code:

Đoạn mã này sử dụng Selenium và BeautifulSoup để thu thập dữ liệu về cầu thủ từ trang web footballtransfers.com, cụ thể là các thông tin về tên, đội bóng, giá trị ước tính (ETV), và chỉ số kỹ năng/tiềm năng từ các bảng trên trang danh sách cầu thủ Premier League. Mã duyệt qua 22 trang, trích xuất dữ liệu từ bảng HTML, lưu vào DataFrame, và xuất ra file CSV. Mục tiêu là tạo tập dữ liệu chứa thông tin chuyển nhượng và hiệu suất của cầu thủ để phân tích hoặc sử dụng trong các mô hình dự đoán.

Vai trò của đoạn mã

Đoạn mã thực hiện việc cào dữ liệu từ trang web footballtransfers.com để thu thập thông tin về cầu thủ Premier League, bao gồm tên, đội bóng, giá trị ước tính (ETV), và chỉ

số kỹ năng/tiềm năng, sau đó lưu vào file **football-transfers-players.csv**. Nó sử dụng Selenium để tải trang web động và BeautifulSoup để phân tích HTML, duyệt qua 22 trang để thu thập dữ liệu đầy đủ. Các bước chi tiết bao gồm:

- **Khởi tạo trình duyệt**: Hàm **setup-driver** cấu hình trình duyệt Chrome bằng **Options**, bật chế độ headless (chạy ẩn), tắt GPU, và sử dụng chế độ no-sandbox để tăng tính ổn định. Hàm trả về đối tượng **driver** để truy cập trang web.

- **Thu thập dữ liệu từ một trang**: Hàm **scrape-page** nhận **driver** và **url**, tải trang bằng **driver.get**, chờ 3 giây để đảm bảo nội dung tải hoàn toàn, sau đó phân tích HTML bằng **BeautifulSoup** với trình phân tích **html.parser**. Tìm bảng HTML với lớp CSS **table table-hover no-cursor table-striped leaguetable mvp-table similar-players-table mb-0**. Nếu không tìm thấy bảng, trả về danh sách rỗng. Với mỗi hàng (**tr**) trong **tbody**:

  - Trích xuất chỉ số **skill** và **pot** từ thẻ **div** với lớp **table-skill–skill** và **table-skill–pot**, chuyển thành số thực bằng **float**, và tạo chuỗi **skill/pot** (ví dụ: "80/90") nếu cả hai giá trị tồn tại, ngược lại gán **None**.
  - Trích xuất tên cầu thủ từ thẻ **span** với lớp **d-none**, gán **None** nếu không tìm thấy.
  - Trích xuất tên đội từ thẻ **span** với lớp **td-team–teamname**, gán **None** nếu không tìm thấy.
  - Trích xuất giá trị ETV từ thẻ **span** với lớp **player-tag**, gán **None** nếu không tìm thấy.
  - Tạo từ điển với các khóa **player-name**, **team**, **price**, **skill/pot**, và thêm vào danh sách **data**.
  - Sử dụng **try-except** để bỏ qua các hàng lỗi, đảm bảo quá trình cào dữ liệu không bị gián đoạn.

  Hàm trả về danh sách **data** chứa các từ điển cho mỗi cầu thủ trên trang.

- **Duyệt qua các trang**: Định nghĩa **base-url** là "https://www.footballtransfers.com/en/players/uk-premier-league" và **total-pages = 22**. Tạo danh sách rỗng **all-data** để lưu toàn bộ dữ liệu. Khởi tạo **driver** bằng **setup-driver**. Dùng vòng lặp để duyệt từ trang 1 đến 22:

  - Với trang 1, sử dụng **base-url**.
  - Với các trang khác, sử dụng URL dạng **{base-url}/{page}**.
  - Gọi **scrape-page** để lấy dữ liệu trang, thêm vào **all-data** bằng **extend**.

  Sử dụng **try-finally** để đảm bảo **driver.quit()** được gọi, đóng trình duyệt và giải phóng tài nguyên.

- **Lưu dữ liệu**: Tạo DataFrame **df-final** từ **all-data**, chứa các cột **player-name**, **team**, **price**, **skill/pot**. Lưu vào file **football-transfers-players.csv** bằng **to-csv** mà không bao gồm chỉ số.

Đoạn mã này cung cấp một quy trình cào dữ liệu hiệu quả, thu thập thông tin chuyển nhượng và hiệu suất của cầu thủ Premier League từ nhiều trang web, tổ chức dữ liệu thành định dạng có cấu trúc, và lưu trữ để sử dụng trong phân tích hoặc dự đoán giá trị chuyển nhượng.

5.2 Kết hợp dataframe của bài 1 để đưa về Dataframe hoàn chỉnh bao gồm cả giá trị chuyển nhượng

5.2.1 Tổng quan về Code:

```
import pandas as pd
import unicodedata
import re

def normalize_name(name):
    if not isinstance(name, str) or name == 'N/A':
        return None
    name = unicodedata.normalize('NFKD', name).encode('ASCII',
        'ignore').decode('ASCII')
    name = name.lower().strip()
    name = re.sub(r'\s+', ' ', name)
    return name

df['normalized_player'] = df['player'].apply(normalize_name)
df_final['normalized_player'] =
    df_final['player_name'].apply(normalize_name)

df_final_subset = df_final[['normalized_player', 'price', 'skill/pot']]

df_merge = df.merge(df_final_subset,
                    on='normalized_player',
                    how='inner')

df_merge['minutes'] = df_merge['minutes'].str.replace(',',
    '').astype(float)
df_merge = df_merge[df_merge['minutes'] > 900]

df_merge = df_merge.drop(columns=['normalized_player'])
cols = [col for col in df_merge.columns if col not in ['price',
    'skill/pot']] + ['price', 'skill/pot']
df_merge = df_merge[cols]

df_merge.to_csv('df_merge.csv', index=False)
```

5.2.2 Giải thích Code:

Giải thích: Đoạn mã bắt đầu bằng việc nhập ba thư viện: - Thư viện pandas (được gán bí danh pd) dùng để xử lý và phân tích dữ liệu dạng bảng. - Thư viện unicodedata hỗ trợ chuẩn hóa các chuỗi ký tự Unicode. - Thư viện re cung cấp các công cụ để xử lý biểu thức chính quy, giúp thao tác với chuỗi.

Hàm Chuẩn Hóa Tên

```
def normalize_name(name):
    if not isinstance(name, str) or name == 'N/A':
```

```
        return None
    name = unicodedata.normalize('NFKD', name).encode('ASCII',
        'ignore').decode('ASCII')
    name = name.lower().strip()
    name = re.sub(r'\s+', ' ', name)
    return name
```

Giải thích: Hàm normalize_name được định nghĩa để chuẩn hóa tên cầu thủ: - Kiểm tra xem đầu vào name có phải là chuỗi và không phải là 'N/A'. Nếu không thỏa mãn, trả về None. - Chuẩn hóa chuỗi Unicode sang dạng NFKD, sau đó chuyển về ASCII, bỏ qua các ký tự không thuộc ASCII (như dấu). - Chuyển chuỗi về chữ thường và loại bỏ khoảng trắng thừa ở đầu/cuối. - Thay thế nhiều khoảng trắng liên tiếp bằng một khoảng trắng duy nhất. - Trả về tên đã chuẩn hóa.

Chuẩn Hóa Tên trong DataFrame

```
df['normalized_player'] = df['player'].apply(normalize_name)
df_final['normalized_player'] =
    df_final['player_name'].apply(normalize_name)
```

Giải thích: - Tạo cột mới normalized_player trong DataFrame df bằng cách áp dụng hàm normalize_name cho cột player. - Tương tự, tạo cột normalized_player trong DataFrame df_final bằng cách áp dụng hàm normalize_name cho cột player_name. Điều này đảm bảo tên cầu thủ trong cả hai DataFrame được chuẩn hóa để dễ dàng so sánh.

Tạo Tập Con Dữ Liệu

```
df_final_subset = df_final[['normalized_player', 'price', 'skill/pot']]
```

Giải thích: - Tạo DataFrame mới df_final_subset từ df_final, chỉ chứa ba cột: normalized_player, price (giá trị cầu thủ), và skill/pot (kỹ năng và tiềm năng). - Bước này giúp giảm số cột, tập trung vào các thông tin cần thiết cho việc ghép dữ liệu.

Ghép Dữ Liệu

```
df_merge = df.merge(df_final_subset,
                    on='normalized_player',
                    how='inner')
```

Giải thích: - Ghép hai DataFrame df và df_final_subset thành df_merge dựa trên cột normalized_player. - Phương thức merge với tham số how='inner' chỉ giữ lại các hàng có giá trị normalized_player xuất hiện ở cả hai DataFrame (giao của hai tập dữ liệu).

Xử Lý và Lọc Dữ Liệu

```
df_merge['minutes'] = df_merge['minutes'].str.replace(',',
    '').astype(float)
df_merge = df_merge[df_merge['minutes'] > 900]
```

Giải thích: - Chuyển đổi cột minutes trong df_merge: loại bỏ dấu phẩy (nếu có) và chuyển dữ liệu sang kiểu số thực (float). - Lọc các hàng trong df_merge, chỉ giữ lại những cầu thủ có số phút thi đấu lớn hơn 900 phút, loại bỏ các cầu thủ ít ra sân.

Sắp Xếp và Xóa Cột

```
df_merge = df_merge.drop(columns=['normalized_player'])
cols = [col for col in df_merge.columns if col not in ['price',
    'skill/pot']] + ['price', 'skill/pot']
df_merge = df_merge[cols]
```

Giải thích: - Xóa cột normalized_player khỏi df_merge vì cột này chỉ dùng để ghép dữ liệu và không cần thiết trong kết quả cuối. - Sắp xếp lại thứ tự cột trong df_merge: đưa các cột price và skill/pot về cuối, các cột còn lại giữ nguyên thứ tự ban đầu. - Gán lại df_merge với thứ tự cột mới.

Lưu Kết Quả

```
df_merge.to_csv('df_merge.csv', index=False)
```

Giải thích: - Lưu DataFrame df_merge vào file CSV có tên df_merge.csv. - Tham số index=False đảm bảo không lưu cột chỉ số (index) của DataFrame vào file CSV.

5.3 Xây dựng mô hình bài toán để dự đoán transfer values các cầu thủ

5.3.1 Tổng quan về Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

def convert_price(price):
    if isinstance(price, str):
        price = price.replace('    ', '').strip()
        if 'M' in price:
            return float(price.replace('M', '')) * 1e6
        elif 'K' in price:
             return float(price.replace('K', '')) * 1e3
```

```
        try:
            return float(price)
        except ValueError:
            return np.nan
    elif isinstance(price, (int, float)):
        return price
    return np.nan

df_merge['price'] = df_merge['price'].apply(convert_price)
df_merge.dropna(subset=['price'], inplace=True)

categorical_cols = df_merge.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = df_merge.select_dtypes(include=np.number).columns.tolist()

cols_to_drop = ['player', 'nationality', 'team', 'position', 'price', 'skill/pot', 'player_name', 'No.']
features_potential = [col for col in numerical_cols if col not in cols_to_drop and col in df_merge.columns]

encoders = {}
for col in ['nationality', 'team', 'position']:
     if col in df_merge.columns:
        le = LabelEncoder()
        df_merge[col] = df_merge[col].astype(str)
        df_merge[col] = le.fit_transform(df_merge[col])
        encoders[col] = le
        if col not in features_potential:
             features_potential.append(col)

final_features = []
for col in features_potential:
    if col in df_merge.columns and pd.api.types.is_numeric_dtype(df_merge[col]):
        final_features.append(col)
    elif col in encoders.keys():
         final_features.append(col)

X = df_merge[final_features]
y = df_merge['price']

X = X.fillna(X.median())

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

model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1, max_depth=15, min_samples_split=5, min_samples_leaf=3)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```python

print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.4f}")

importances = model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': final_features, 'Importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 5))
plt.scatter(y_test / 1e6, y_pred / 1e6, alpha=0.6, edgecolors='k', s=50)
plt.plot([min(y_test / 1e6), max(y_test / 1e6)], [min(y_test / 1e6), max(y_test / 1e6)], '--', color='red', lw=2)

plt.title('Predicted Transfer Values')
plt.xlabel('Truth Label')
plt.ylabel('Predicted Label')
plt.grid(True)
plt.tight_layout()
plt.savefig('actual_vs_predicted.png', dpi=300, format='png', bbox_inches='tight')
plt.show()
```

5.3.2 Mô tả và giải thích vai trò đoạn mã

```python
def convert_price(price):
    if isinstance(price, str):
        price = price.replace('€', '').strip()
        if 'M' in price:
            return float(price.replace('M', '')) * 1e6
        elif 'K' in price:
            return float(price.replace('K', '')) * 1e3
        try:
            return float(price)
        except ValueError:
            return np.nan
    elif isinstance(price, (int, float)):
        return price
    return np.nan
```

Giải thích: Hàm `convert_price` chuyển đổi giá trị giá (có thể ở dạng chuỗi như "1M€" hoặc "500K€") thành số thực (đơn vị euro):

- Nếu giá là chuỗi, loại bỏ ký hiệu "€" và khoảng trắng.
- Nếu có "M", nhân với 1 triệu (1e6); nếu có "K", nhân với 1 nghìn (1e3).
- Thử chuyển chuỗi thành số thực, nếu không được thì trả về `NaN`.
- Nếu giá đã là số (int hoặc float), giữ nguyên.

- Các trường hợp khác trả về NaN (giá trị không hợp lệ).

```
df_merge['price'] = df_merge['price'].apply(convert_price)
df_merge.dropna(subset=['price'], inplace=True)
```

Giải thích:

- Áp dụng hàm `convert_price` cho cột "price" trong DataFrame `df_merge` để chuyển tất cả giá trị thành số thực.
- Xóa các hàng có giá trị NaN trong cột "price" để đảm bảo dữ liệu sạch.

```
categorical_cols = df_merge.select_dtypes(include=['object',
    'category']).columns.tolist()
numerical_cols =
    df_merge.select_dtypes(include=np.number).columns.tolist()
```

Giải thích:

- `categorical_cols`: Lấy danh sách các cột có kiểu dữ liệu là chuỗi hoặc danh mục (object, category).
- `numerical_cols`: Lấy danh sách các cột có kiểu dữ liệu số (int, float).

```
cols_to_drop = ['player', 'nationality', 'team', 'position', 'price',
    'skill/pot', 'player_name', 'No.']
features_potential = [col for col in numerical_cols if col not in
    cols_to_drop and col in df_merge.columns]
```

Giải thích:

- `cols_to_drop`: Liệt kê các cột không dùng làm đặc trưng (feature), ví dụ: thông tin định danh như tên cầu thủ, quốc tịch, hoặc giá (vì giá là mục tiêu dự đoán).
- `features_potential`: Tạo danh sách các cột số có thể dùng làm đặc trưng, loại bỏ các cột trong `cols_to_drop` và đảm bảo cột tồn tại trong DataFrame.

```
encoders = {}
for col in ['nationality', 'team', 'position']:
     if col in df_merge.columns:
        le = LabelEncoder()
        df_merge[col] = df_merge[col].astype(str)
        df_merge[col] = le.fit_transform(df_merge[col])
        encoders[col] = le
        if col not in features_potential:
             features_potential.append(col)
```

Giải thích:

- Tạo từ điển `encoders` để lưu các bộ mã hóa `LabelEncoder` cho các cột danh mục.

- Đối với các cột "nationality", "team", "position":
  - Chuyển dữ liệu thành chuỗi (để đảm bảo đồng nhất).
  - Sử dụng `LabelEncoder` để mã hóa các giá trị danh mục thành số (ví dụ: "Brazil" thành 0, "Spain" thành 1).
  - Lưu bộ mã hóa vào `encoders`.
  - Thêm cột đã mã hóa vào `features_potential` nếu chưa có.

```
final_features = []
for col in features_potential:
    if col in df_merge.columns and
       pd.api.types.is_numeric_dtype(df_merge[col]):
        final_features.append(col)
    elif col in encoders.keys():
         final_features.append(col)
```

Giải thích:

- Tạo danh sách `final_features` chứa các đặc trưng cuối cùng:
  - Các cột số (kiểu `numeric`) trong `features_potential`.
  - Các cột danh mục đã được mã hóa (có trong `encoders`).
- Đảm bảo chỉ chọn các cột tồn tại trong DataFrame.

```
X = df_merge[final_features]
y = df_merge['price']
X = X.fillna(X.median())
```

Giải thích:

- `X`: Ma trận đặc trưng, chứa các cột trong `final_features`.
- `y`: Biến mục tiêu, là cột "price" (giá trị chuyển nhượng).
- Điền giá trị thiếu trong `X` bằng giá trị trung vị (median) của từng cột để đảm bảo không có dữ liệu thiếu.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

Giải thích:

- Chia dữ liệu thành tập huấn luyện (80%) và tập kiểm tra (20%) bằng `train_test_split`.
- `random_state=42` đảm bảo việc chia dữ liệu là cố định, giúp tái lập kết quả.

```
model = RandomForestRegressor(n_estimators=100, random_state=42,
    n_jobs=-1, max_depth=15, min_samples_split=5, min_samples_leaf=3)
model.fit(X_train, y_train)
```

Giải thích:

- Khởi tạo mô hình `RandomForestRegressor` với các tham số:
  - `n_estimators=100`: Sử dụng 100 cây quyết định.
  - `random_state=42`: Đảm bảo kết quả cố định.
  - `n_jobs=-1`: Sử dụng tất cả CPU để tăng tốc.
  - `max_depth=15`, `min_samples_split=5`, `min_samples_leaf=3`: Giới hạn độ sâu và kích thước node để tránh overfitting.
- Huấn luyện mô hình trên tập `X_train`, `y_train`.

```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R^2 Score: {r2:.4f}")
```

Giải thích:

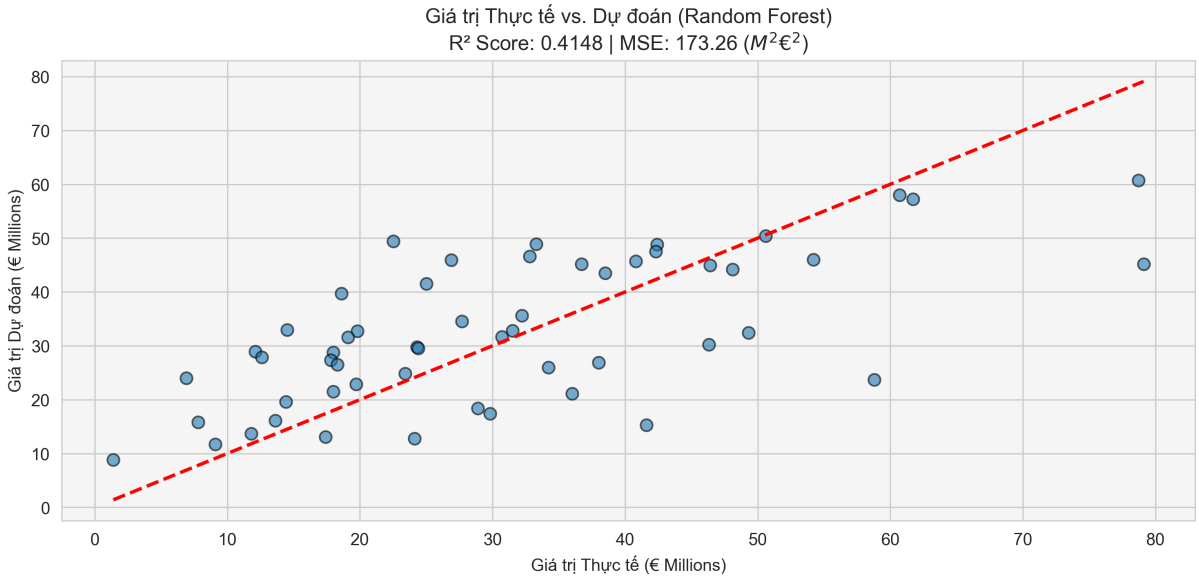
- `y_pred`: Dự đoán giá trị chuyển nhượng trên tập kiểm tra `X_test`.
- Tính `mse` (Mean Squared Error): Sai số bình phương trung bình, đo độ lệch giữa giá trị thực và dự đoán.
- Tính `r2` (R^2 Score): Hệ số xác định, đo mức độ mô hình giải thích được biến động của dữ liệu (gần 1 là tốt).
- In kết quả với định dạng số thập phân.

```
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': final_features,
    'Importance': importances})
feature_importance_df =
    feature_importance_df.sort_values(by='Importance', ascending=False)
```

Giải thích:

- `importances`: Lấy tầm quan trọng của từng đặc trưng từ mô hình.
- Tạo DataFrame `feature_importance_df` để lưu tên đặc trưng và tầm quan trọng.
- Sắp xếp theo tầm quan trọng giảm dần để biết đặc trưng nào ảnh hưởng lớn nhất đến dự đoán.

5.3.3 Kết quả thử nghiệm



6 KẾT QUẢ THỬ NGHIỆM

6.1 CHƯƠNG 1: Kết quả thu được là File CSV chứa đầy đủ các thuộc tính của cầu thủ trong mùa giải 2024 - 2025 Premier League

1			Nation	Team	Position	Age	Playing Time		
2	No.	Player	nationality	team	position	birth_year	games	games_starts	minutes
3	0	Aaron Cresswell	ENG	West Ham	DF	36	14	7	589
4	1	Aaron Ramsdale	ENG	Southampton	GK	27	26	26	2,340
5	2	Aaron Wan-Bissaka	ENG	West Ham	DF	28	32	31	2,794
6	3	Abdoulaye Doucouré	MLI	Everton	MF	32	30	29	2,425
7	4	Abdukodir Khusanov	UZB	Manchester City	DF	21	6	6	503
8	5	Abdul Fatawu Issahaku	GHA	Leicester City	FW	21	11	6	579
9	6	Adam Armstrong	ENG	Southampton	FW,MF	28	20	15	1,248
10	7	Adam Lallana	ENG	Southampton	MF	37	14	5	361
11	8	Adam Smith	ENG	Bournemouth	DF	34	22	17	1,409
12	9	Adam Webster	ENG	Brighton	DF	30	11	8	617
13	10	Adam Wharton	ENG	Crystal Palace	MF	21	19	15	1,258
14	11	Adama Traoré	ESP	Fulham	FW,MF	29	32	16	1,568
15	12	Albert Grønbaek	DEN	Southampton	FW,MF	24	4	2	143
16	13	Alejandro Garnacho	ARG	Manchester Utd	MF,FW	21	33	22	2,056
17	14	Alex Iwobi	NGA	Fulham	FW,MF	29	34	32	2,721
18	15	Alex McCarthy	ENG	Southampton	GK	36	5	5	450
19	16	Alex Palmer	ENG	Ipswich Town	GK	29	10	10	900
20	17	Alex Scott	ENG	Bournemouth	MF	22	17	7	683
21	18	Alexander Isak	SWE	Newcastle Utd	FW	26	31	31	2,487
22	19	Alexis Mac Allister	ARG	Liverpool	MF	27	33	30	2,553

6.2 CHƯƠNG 2: Cập nhật, thống kê dữ liệu

6.2.1 Top 3 cầu thủ có chỉ số cao nhất và thấp nhất ứng với mỗi tiêu chí

Bảng 1: Thống kê theo năm sinh (Tuổi)

Top 3 cao nhất (Già nhất)			**Top 3 thấp nhất (Trẻ nhất)**		
Hạng	**Cầu thủ**	**Tuổi**	**Hạng**	**Cầu thủ**	**Tuổi**
1	Łukasz Fabiański	40	1	Mikey Moore	18
2	Ashley Young	40	2	Ethan Nwaneri	18
3	James Milner	39	3	Harry Amass	18

Bảng 2: Thống kê theo số trận thi đấu

Top 3 cao nhất			**Top 3 thấp nhất**		
Hạng	**Cầu thủ**	**Số trận**	**Hạng**	**Cầu thủ**	**Số trận**
1	Virgil van Dijk	34	1	Altay Bayındır	1
2	Rayan Aït-Nouri	34	2	Tyler Fredricson	1
3	Tyrick Mitchell	34	3	Brandon Austin	1

Bảng 3: Thống kê theo số trận đá chính

Top 3 cao nhất			**Top 3 thấp nhất**		
Hạng	**Cầu thủ**	**Số trận**	**Hạng**	**Cầu thủ**	**Số trận**
1	Virgil van Dijk	34	1	Ali Al Hamadi	0
2	Tyrick Mitchell	34	2	Toby Collyer	0
3	Rayan Aït-Nouri	34	3	William Osula	0

Bảng 4: Thống kê theo số phút thi đấu

Top 3 cao nhất			**Top 3 thấp nhất**		
Hạng	**Cầu thủ**	**Phút**	**Hạng**	**Cầu thủ**	**Phút**
1	Omar Marmoush	998.0	1	İlkay Gündoğan	0.0
2	Christian Eriksen	994.0	2	Mateo Kovačić	0.0
3	Paul Onuachu	984.0	3	André Onana	0.0

Bảng 5: Thống kê theo số bàn thắng

Top 3 cao nhất			Top 3 thấp nhất		
Hạng	**Cầu thủ**	**Bàn thắng**	**Hạng**	**Cầu thủ**	**Bàn thắng**
1	Mohamed Salah	28	1	Woyo Coulibaly	0
2	Alexander Isak	22	2	Arijanet Muric	0
3	Erling Haaland	21	3	Armando Broja	0

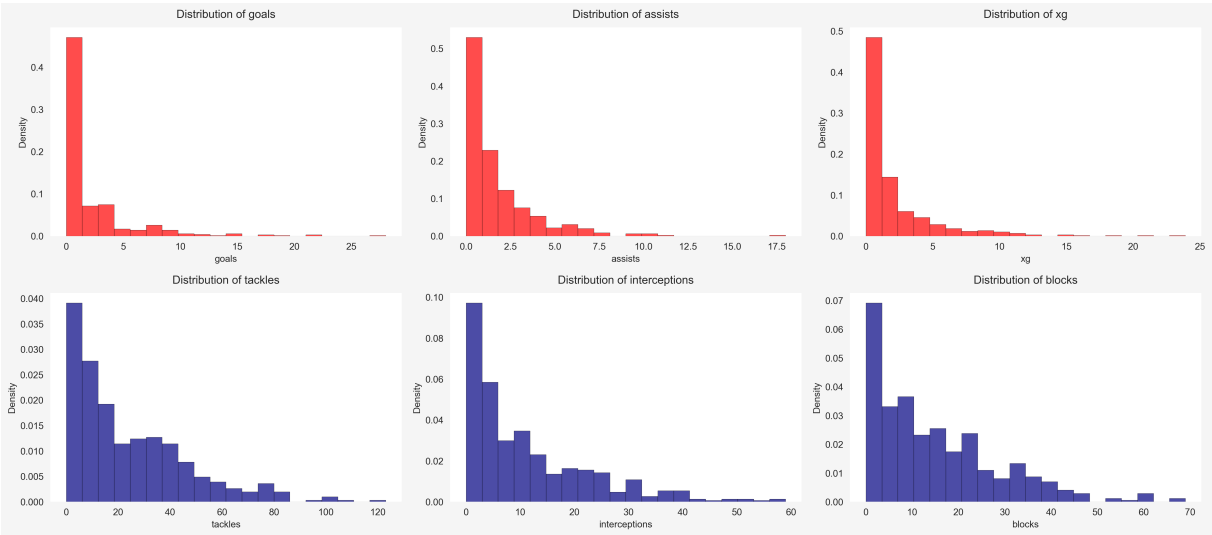
Bảng 6: Thống kê theo số kiến tạo

Top 3 cao nhất			Top 3 thấp nhất		
Hạng	**Cầu thủ**	**Kiến tạo**	**Hạng**	**Cầu thủ**	**Kiến tạo**
1	Mohamed Salah	18	1	Aaron Cresswell	0
2	Jacob Murphy	11	2	Aaron Ramsdale	0
3	Bukayo Saka	10	3	Ali Al Hamadi	0

6.2.2 Tính toán giá trị trung bình, trung vị, độ lệch chuẩn của từng tiêu chí

		Median	Mean	Std	Median	Mean	Std	Median	Mean
1		birth_year	birth_year	birth_year	games	games	games	games_starts	games_starts
2	all	27.0	27.004008016032063	4.240271640150538	22.0	20.482965931863728	9.932362140579054	14.0	15.026052104208416
3	West Ham	28.0	28.84	4.938623289946299	20.0	20.92	8.281102986116442	14.0	14.96
4	Southampton	27.0	27.0	4.27617987059879	20.0	18.137931034482758	10.056000830815378	13.0	12.862068965517242
5	Everton	26.0	27.782608695652176	5.1430139884483586	22.0	20.82608695652174	9.943715118391678	14.0	16.260869565217739
6	Manchester City	27.0	27.12	4.969238707622459	23.0	19.88	8.973665174646682	17.0	15.36
7	Leicester City	27.0	27.384615384615383	4.364189941576082	21.0	19.73076923076923	9.568940139044418	14.5	14.384615384615385
8	Bournemouth	26.0	26.458333333333332	3.764585162972856	24.5	20.833333333333332	10.344780942756747	16.5	15.583333333333334
9	Brighton	25.0	26.142857142857142	5.169098250137709	20.0	18.964285714285715	9.758119668814247	9.0	13.357142857142858
10	Crystal Palace	27.0	27.130434782608695	3.251633824655579	25.0	21.434782608695652	11.676807187196188	15.0	16.260869565217739
11	Fulham	29.0	28.90909090909091	3.2646605731696914	26.0	23.772727272727273	9.211441220061568	17.0	16.954545454545453
12	Manchester Utd	26.0	25.79310344827586	4.8871501361429415	20.0	17.551724137931036	11.5188857561849	12.0	12.89655172413793
13	Ipswich Town	27.0	27.033333333333335	3.295590924285034	18.0	17.666666666666668	8.742813140471172	11.0	12.466666666666667
14	Newcastle Utd	28.0	28.130434782608695	4.673954407391191	27.0	22.565217391304348	9.917047246381179	13.0	16.260869565217739
15	Liverpool	27.0	27.333333333333332	3.719318934070233	28.0	24.476190476190474	8.902915520317196	19.0	17.80952380952381
16	Aston Villa	28.0	27.285714285714285	4.035556254807296	20.0	18.857142857142858	9.965549122460303	9.5	13.357142857142858
17	Wolves	27.5	27.916666666666668	3.9882072540741005	25.0	21.875	9.79490772755437	14.0	16.0
18	Nott'ham Forest	27.5	27.5	3.595632271043777	29.5	23.863636363636363	10.575332729406783	18.5	17.0
19	Tottenham	26.0	25.821428571428573	4.472579597275856	19.5	18.25	9.713011509958823	14.5	13.357142857142858
20	Chelsea	24.0	24.307692307692307	2.54195563720897	17.5	19.153846153846153	10.880045248774687	11.5	14.384615384615385
21	Brentford	25.0	26.333333333333332	3.890158522905376	27.0	22.857142857142858	11.145787160563017	21.0	17.80952380952381
22	Arsenal	27.5	26.727272727272727	3.9542842128966447	22.5	22.59090909090909	7.926201822100881	16.0	17.0

6.2.3 Trực quan hoá dữ liệu cho các tiêu chí

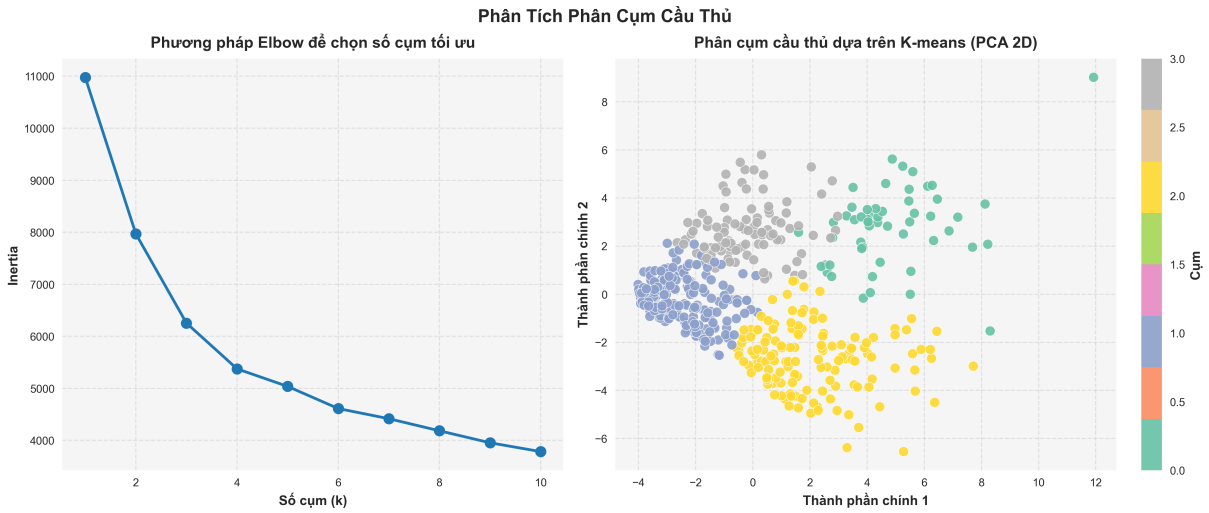


6.2.4 Thống kê các Team có chỉ số cao nhất theo từng tiêu chí

Bảng 7: Top Team Statistics

Statistic	**Top Team**	**Value**
Minutes	Ipswich Town	8635.0
Goals	Liverpool	79
Assists	Liverpool	59
Yellow Cards	Chelsea	94
Red Cards	Arsenal, Ipswich Town	5
xG (Expected Goals)	Liverpool	76.2
xG Assist	Liverpool	55.2
Progressive Carries	Manchester City	1041

6.3 CHƯƠNG 3: Phân cụm dữ liệu cho mục đích phân loại

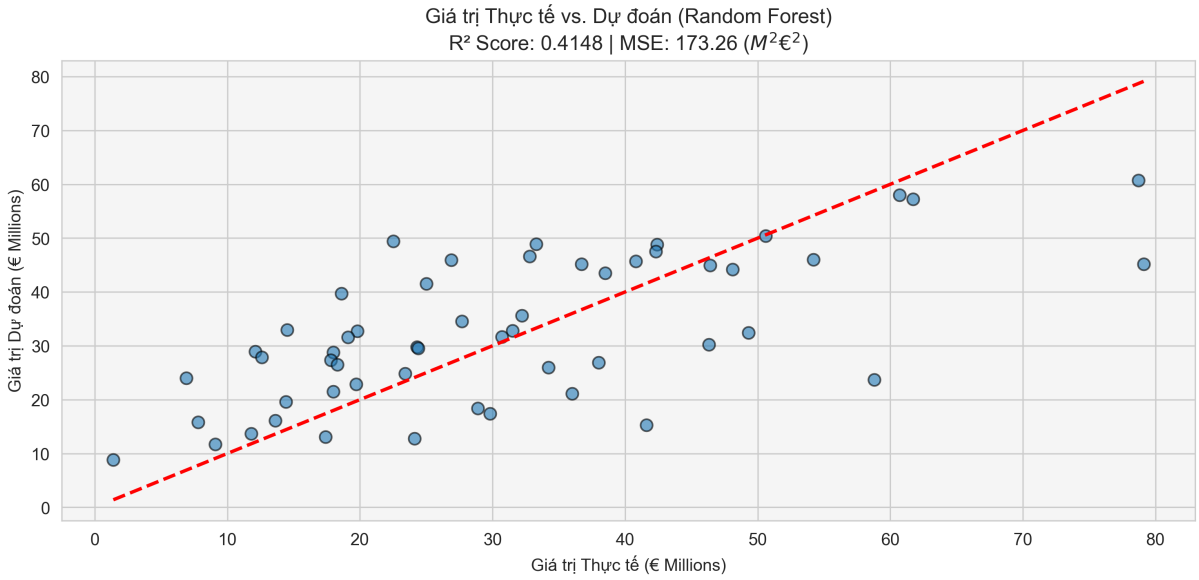


6.4 CHƯƠNG 4: Thu thập dữ liệu và dự đoán giá trị chuyển nhượng

6.4.1 Thu thập dữ liệu

	player_name	team	price	skill/pot
484	Joe Lumley	Southampton	€0.7M	58.4/58.4
485	John Ruddy	Newcastle Utd.	€0.7M	60.5/60.5
486	Séamus Coleman	Everton	€0.7M	59.8/59.8
487	Harry Amass	Man Utd	€0.7M	43.8/63.6
488	Jakub Stolarczyk	Leicester	€0.7M	52.8/54.8
489	Fraser Forster	Tottenham	€0.7M	52.4/52.4
490	Joel Ward	C. Palace	€0.7M	58.1/58.1
491	Harvey Davies	Liverpool	€0.6M	54.1/68.8
492	Alex McCarthy	Southampton	€0.6M	52.2/52.2
493	Tom King	Wolverhampton	€0.6M	51.1/51.1
494	Lucas Bergström	Chelsea	€0.6M	52.4/60.6
495	João Virgínia	Everton	€0.6M	51.7/54.4
496	Tom McGill	Brighton	€0.5M	49.8/53.1
497	Massimo Luongo	Ipswich Town	€0.5M	56.7/56.7
498	Alfie Whiteman	Tottenham	€0.5M	49.1/53.1
499	Remi Matthews	C. Palace	€0.4M	47.7/47.8
500	Mark Gillespie	Newcastle Utd.	€0.4M	43.7/44.7
501	Wayne Hennessey	Nottingham	€0.4M	45.4/45.4
502	Scott Carson	Man City	€0.4M	44.5/44.9
503	Tom Heaton	Man Utd	€0.4M	46.8/46.8
504	Ranel Young	Liverpool	€0.2M	43.6/60.6
505	Rento Takaoka	Southampton	€0.2M	32.6/32.6
506	Richard Olise	Chelsea	€90K	47.0/57.5

6.4.2 Xây dựng mô hình dự đoán



Kết luận

Bài tập lớn này đã cung cấp một cơ hội toàn diện để khám phá việc thu thập, phân tích và trực quan hóa dữ liệu thống kê của các cầu thủ trong mùa giải Ngoại hạng Anh 2024-2025. Bằng cách sử dụng lập trình Python và các nguồn dữ liệu bên ngoài như Foref.com và FootballTransfers.com, chúng tôi đã hoàn thành các yêu cầu được đề ra trong bài tập. Những thành tựu chính bao gồm:

1. **Thu thập và lưu trữ dữ liệu**: Chúng tôi đã phát triển một chương trình Python để thu thập và tổng hợp dữ liệu thống kê chi tiết cho các cầu thủ có thời gian thi đấu trên 90 phút, lưu kết quả vào tệp `results.csv` với cấu trúc rõ ràng. Dữ liệu không khả dụng được đánh dấu chính xác là "N/a".

2. **Phân tích thống kê**: Việc xác định ba cầu thủ có thành tích cao nhất và thấp nhất cho mỗi chỉ số thống kê, cùng với tính toán trung vị, trung bình và độ lệch chuẩn cho toàn bộ cầu thủ và từng đội, đã mang lại những hiểu biết giá trị về hiệu suất cá nhân và tập thể. Kết quả được lưu trữ trong các tệp `top_3.txt` và `results2.csv`, hỗ trợ phân tích sâu hơn.

3. **Trực quan hóa**: Các biểu đồ histogram được tạo ra cho từng chỉ số thống kê đã cung cấp một cái nhìn trực quan về phân bố dữ liệu trên toàn giải đấu và trong từng đội. Những biểu đồ này giúp nhận diện xu hướng và sự chênh lệch trong hiệu suất của cầu thủ, nâng cao tính dễ hiểu của dữ liệu.

4. **Đánh giá hiệu suất đội bóng**: Phân tích các đội có điểm số cao nhất cho mỗi chỉ số thống kê đã mang lại cái nhìn sâu sắc về những đội xuất sắc trong các lĩnh vực cụ thể. Dựa trên phân tích này, chúng tôi đã đề xuất một đánh giá dựa trên dữ liệu về đội bóng có thành tích tốt nhất trong mùa giải 2024-2025, được hỗ trợ bởi bằng chứng định lượng.

5. **Phân cụm và giảm chiều dữ liệu**: Việc áp dụng thuật toán K-means đã cho phép phân loại cầu thủ thành các nhóm có ý nghĩa dựa trên hồ sơ thống kê của họ. Số lượng cụm tối ưu được xác định thông qua đánh giá cẩn thận, và việc sử dụng PCA để giảm chiều dữ liệu xuống 2D đã cho phép trực quan hóa hiệu quả các cụm cầu thủ, làm nổi bật các mô hình tiềm ẩn trong hiệu suất.

6. **Định giá cầu thủ**: Bằng cách thu thập giá trị chuyển nhượng cho các cầu thủ có thời gian thi đấu trên 900 phút và đề xuất phương pháp chọn đặc trưng cùng mô hình, chúng tôi đã giải quyết bài toán ước lượng giá trị cầu thủ. Phân tích này đã làm sáng tỏ mối liên hệ giữa các chỉ số hiệu suất và giá trị thị trường, cung cấp một khung phân tích chắc chắn cho các nghiên cứu trong tương lai.

Dự án không chỉ rèn luyện các kỹ năng kỹ thuật trong lập trình Python, thu thập dữ liệu, phân tích thống kê và học máy mà còn nhấn mạnh tầm quan trọng của việc ra quyết định dựa trên dữ liệu trong phân tích thể thao. Các phương pháp và hiểu biết thu được từ công việc này có thể được mở rộng sang các môn thể thao hoặc mùa giải khác, mang lại một cách tiếp cận có khả năng mở rộng cho việc đánh giá hiệu suất. Trong tương lai, việc tích hợp cập nhật dữ liệu theo thời gian thực hoặc các mô hình dự đoán tiên tiến có thể nâng cao hơn nữa độ sâu và tính ứng dụng của phân tích này. Nhìn chung, bài tập lớn này là một cơ hội quý giá để kết nối chuyên môn lập trình với phân tích dữ liệu thể thao thực tế, mang lại những kết quả có ý nghĩa và khả năng ứng dụng cao.