HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN I

BÁO CÁO BÀI TẬP LỚN

Môn Python

Giảng viên hướng dẫn: **Ts. Kim Ngọc Bách**
Sinh viên: **Nguyễn Minh Hiếu**
Mã sinh viên: **B23DCCE030**
Lớp: **D23CQCE06-B**

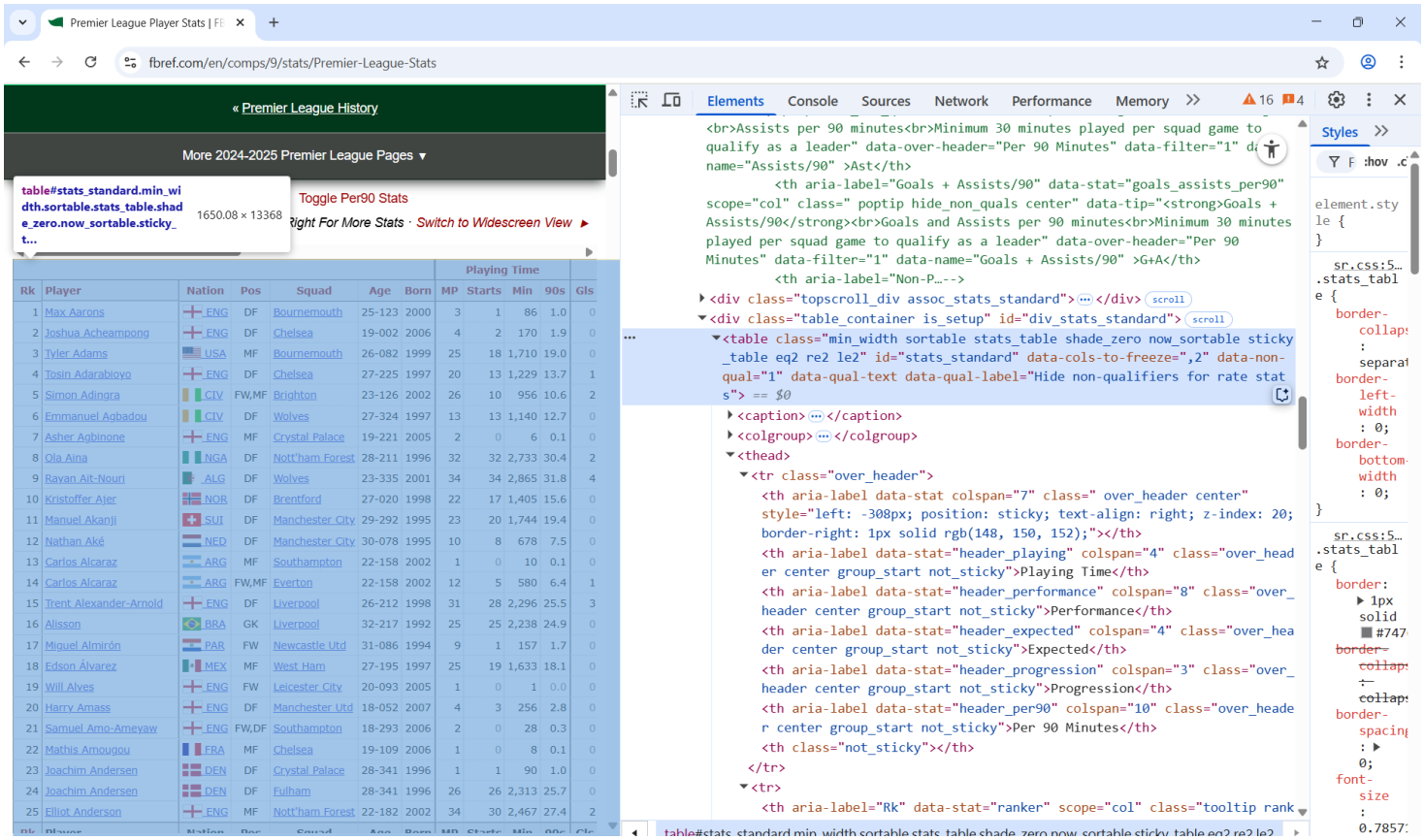
Mục Lục

I. (BAI1.py): Viết chương trình Python thu thập dữ liệu cầu thủ epl

1. Ý tưởng thực hiện :

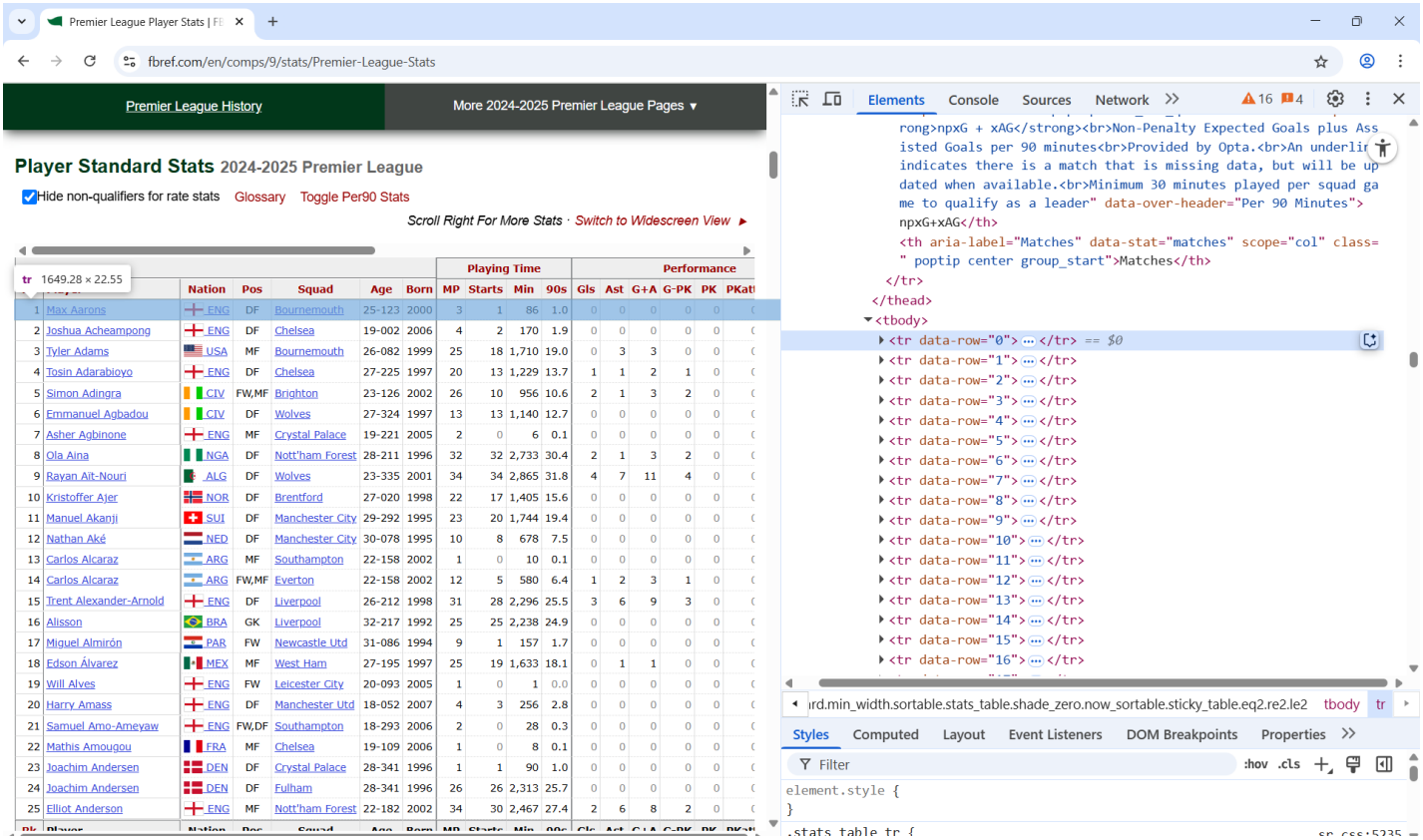
- Truy cập vào các link url lưu trữ thông tin, nhấm f12 để bật DevTools, chọn tab Element, xác định vị trí DOM của mục dữ liệu cần scrape.
- Ví dụ để lấy thông tin chung của các cầu thủ, ta truy cập web : https://fbref.com/en/comps/9/stats/Premier-League-Stats
- Ta cần lấy bảng thông tin của các cầu thủ, ta cần trỏ vào thẻ table có id = "stats_standard".

(Mô tả ở ảnh)



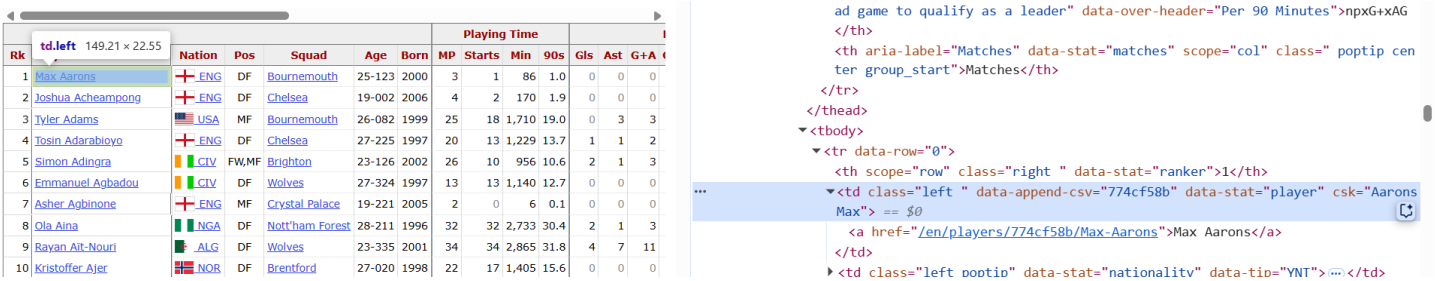
- Sau khi lấy được bảng, ta duyệt từng hàng để lấy thông tin của các cầu thủ, chính là các thẻ <tr> nằm trong thẻ <tbody> của table

(Mô tả ở ảnh)



- Bước cuối cùng là lấy chỉ số của từng cầu thủ, trong từng thẻ <tr>, tìm tới thẻ <td> và lấy thuộc tính data-stat (Ví dụ như name thì data-stat = “player”.

(Mô tả ở ảnh)



2. Trình bày các cài đặt bằng ngôn ngữ Python :

- Trước hết, ta lưu url, table_id, data-stat của các features vào list :

(Ảnh dưới đây là mô tả các thông tin trang url thứ nhất, 7 trang còn lại tương tự đã được trình bày trong code)

```
PAGES = [
    #General
    {
        'url'               :       'https://fbref.com/en/comps/9/stats/Premier-League-Stats',
        'table_id'          :       'stats_standard',
        'filter'            :       'minutes',
        'stats': {
            'Position'      :       'position',
            'Age'           :       'birth_year',
            #Playing Time
            'Matches_played':       'games',
            'Starts'        :       'games_starts',
            'Minutes'       :       'minutes',
            #Performance
            'Goals'         :       'goals',
            'Assists'       :       'assists',
            'Yellow Cards'  :       'cards_yellow',
            'Red Cards'     :       'cards_red',
            #Expected
            'Expected_xG'   :       'xg',
            'Expected_xAG'  :       'xg_assist',
            #Progression
            'PrgC'          :       'progressive_carries',
            'PrgP'          :       'progressive_passes',
            'PrgR'          :       'progressive_passes_received',
            #Per 90 minutes
            'Per90_Gls'     :       'goals_per90',
            'Per90_Ast'     :       'assists_per90',
            'Per90_xG90'    :       'xg_per90',
            'Per90_xAG'     :       'xg_assist_per90',
        }
    },
```

- Sau khi lưu thông tin xong, sử dụng selenium để khởi chạy trình duyệt Chrome, điều hướng tới url, render trang và lấy về HTML thô thông qua driver.page_source, ta sẽ dùng BeautifulSoup để: Parse HTML thành cây DOM, chuyển chuỗi HTML thành đối tượng soup có cấu trúc phân cấp, dễ duyệt, tốc độ cao.
- Trước hết, ta cần truy cập vào thông tin chung (page đầu tiên) để lấy những cầu thủ có thời gian thi đấu > 90 phút.
- Dùng cấu trúc dữ liệu dict để lưu trữ, với dict player, ta lưu trữ cặp key (Name, Team) (do một mùa giải một cầu thủ có thể thi đấu cho 2 đội) và value là giá trị của các cột.
- Dùng thêm 1 set satisfy_player để đánh dấu các cầu thủ chơi trên 90 phút (do 7 link url sau không có cột Minutes nên duyệt url đầu rồi lưu luôn, dùng set để dễ dàng truy vấn)

```
import time
from bs4 import BeautifulSoup
from selenium import webdriver
driver = webdriver.Chrome()
# Lấy Page[0] để xử lý các thông tin chung trước
std = PAGES[0]
driver.get(std['url'])
time.sleep(4)
soup0 = BeautifulSoup(driver.page_source, 'html.parser')
table0 = soup0.find('table', {'id': std['table_id']}).find('tbody')

players = {}            # key = (Name, Team) → dict chứa tất cả stats
satisfy_player = set()   # tập (Name, Team) đã chơi > 90'
```

- Định nghĩa hàm get_data để lấy thông tin data-stat

```
def get_data(row, data_stat):
    cell = row.find('td', {'data-stat': data_stat})
    return cell.get_text(strip=True) if cell and cell.get_text(strip=True) else 'N/a'

for row in table0.find_all('tr'):
    # loại bỏ dòng tiêu đề
    if 'thread' in row.get('class', []):
        continue
    #check thời gian thi đấu
    mn = get_data(row, std['filter'])
    if mn == 'N/a' or int(mn.replace(',', '')) <= 90:
        continue
    name = get_data(row, 'player')
    team = get_data(row, 'team')
    key  = (name, team)
    satisfy_player.add(key)
    # Quốc tịch
    nat_cell = row.find('td', {'data-stat': 'nationality'})
    span = nat_cell.find('span') if nat_cell else None
    nation = span.contents[-1].strip() if span and span.contents else 'N/a'
    # Khởi tạo dict với tất cả cột của stats_standard
    players[key] = {'Name': name, 'Team': team, 'Nation': nation}
    for col, ds in std['stats'].items():
        val = get_data(row, ds)
        # Nếu col là Age mà file trả về birth_year, thì tính ngược
        if col == 'Age' and val.isdigit():
            players[key][col] = str(2025 - int(val))
        else:
            players[key][col] = val
```

- Sau khi cập nhật được các cầu thủ thỏa mãn điều kiện vào set, 7 page sau ta chỉ cần lấy các thông tin, check xem cầu thủ đó có thỏa mãn hay không rồi lưu vào dict .

```
#. VÒNG 2: Duyệt tiếp các page còn lại, chỉ cập nhật nếu key trong satisfy_player ===
for page in PAGES[1:]:
    driver.get(page['url'])
    time.sleep(4)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    body = soup.find('table', {'id': page['table_id']}).find('tbody')

    for row in body.find_all('tr'):
        if 'thread' in row.get('class', []):
            continue

        name = get_data(row, 'player')
        team = get_data(row, 'team')
        key  = (name, team)
        #kiểm tra key có nằm trong set hay không
        if key not in satisfy_player:
            continue

        # Cập nhật thêm các cột stats của page này
        for col, ds in page['stats'].items():
            players[key][col] = get_data(row, ds)

driver.quit()
```

- Cuối cùng ghi dữ liệu vào result.csv sort theo tên thứ tự từ điển, các ô trống sẽ điền N/a

```
# === Xuất file CSV với BOM UTF-8 và sắp theo Name → Team ===
fieldnames = ['Name', 'Team', 'Nation'] + [
    c for p in PAGES for c in p['stats'].keys()
]


import csv
with open('results.csv', 'w', newline='', encoding='utf-8-sig') as f:
    writer = csv.DictWriter(f, fieldnames=fieldnames, restval='N/a')
    writer.writeheader()
    for pl in sorted(players.values(), key=lambda x: (x['Name'].lower(), x['Team'].lower())):
        writer.writerow(pl)
```

II. Bài 2 : Xử lý các yêu cầu về chỉ số

1. (BAI2_1.py) : Tìm top 3 của các chỉ số

- Import các thư viện cần dùng
- Đọc dữ liệu và chuẩn hóa dữ liệu các giá trị N/a thành giá trị nan trong numpy, chỉ giữ lại các cột là số

```
import pandas as pd
import numpy as np


df = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv', encoding='utf-8-sig', thousands=',')
df.replace('N/a', np.nan, inplace=True)
stats = [c for c in df.columns if c not in ['Name', 'Team', 'Nation', 'Position']]

for stat in stats:
    df[stat] = pd.to_numeric(df[stat].astype(str).str.replace(',', ''),  errors='coerce')
```

- Duyệt từng cột và lấy top 3 chỉ số max, min ra file top_3.txt
- Đối với các giá trị N/a ta drop để không bị ảnh hưởng tới kết quả

```
with open('top_3.txt', 'w', encoding='utf-8') as f:
    for stat in stats:
        sub = df[['Name', 'Team', 'Nation', 'Position', stat]].dropna(subset=[stat])
        f.write(f'--- {stat} ---\n')
        if sub.empty:
            f.write('Không có dữ liệu\n\n')
            continue
        bottom3 = sub.nsmallest(3, stat)
        top3    = sub.nlargest(3, stat)
        f.write('Top 3 lowest:\n')
        for _, r in bottom3.iterrows():
            f.write(f"  {r['Name']} trong đội {r['Team']} : {r[stat]}\n")
        f.write('Top 3 highest:\n')
        for _, r in top3.iterrows():
            f.write(f"  {r['Name']} trong đội {r['Team']} : {r[stat]}\n")
        f.write('\n')
```

2. (BAI2_2.py) : Tìm mean, median, std

- Import các thư viện cần dùng
- Đọc dữ liệu và chuẩn hóa dữ liệu các giá trị N/a thành giá trị nan trong numpy đồng thời chuẩn bị các hàng và các cột cần dùng

```
import pandas as pd
import numpy as np


df = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv', encoding='utf-8-sig', thousands=',')
df.replace('N/a', np.nan, inplace=True)
exclude = ['Name','Nation','Position']
stats   = [c for c in df.columns if c not in exclude + ['Team']]
for stat in stats:
    df[stat] = pd.to_numeric(df[stat].astype(str).str.replace(',', ''), errors='coerce')
scopes = ['All'] + df['Team'].dropna().unique().tolist()
cols = ['Teams']
for stat in stats:
    cols += [f'Median of {stat}', f'Mean of {stat}', f'Std of {stat}']
```

- Duyệt từng hàng và lần lượt tìm mean, median, std bằng các method có sẵn
- Lưu vào resulsts2.csv

```
17  rows = []
18  for scope in scopes:
19      row = {'Teams': scope}
20      if scope == 'All':
21          subdf = df
22      else:
23          subdf = df[df['Team'] == scope]
24      for stat in stats:
25          arr = subdf[stat].dropna()
26          row[f'Median of {stat}'] = arr.median()
27          row[f'Mean of {stat}']   = arr.mean()
28          row[f'Std of {stat}']    = arr.std()
29      rows.append(row)
30  res2 = pd.DataFrame(rows, columns=cols)
31  res2.to_csv('results2.csv', index=False, encoding='utf-8-sig')
```

3. (BAI2_3.py): Plot a histogram

- Import các thư viện cần dùng
- Tạo thư mục đồng thời đọc và chuẩn hóa dữ liệu

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

plots_dir  = Path(__file__).parent / 'plots'
plots_dir.mkdir(exist_ok=True)
#Đọc và chuẩn hóa dữ liệu
df = pd.read_csv(r'C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv', encoding='utf-8-sig', thousands=',')
df.replace('N/a', np.nan, inplace=True)
```

- Chọn 3 chỉ số tấn công là Goals, Assists, GCA và 3 chỉ số phòng thủ Tkl, Sh, Int

```
attack_stats  = ['Goals', 'Assists', 'GCA']
defense_stats = ['Tkl', 'Sh', 'Int']
plot_stats    = [s for s in (attack_stats + defense_stats) if s in df.columns]
# Ép kiểu numeric
for stat in plot_stats:
    df[stat] = pd.to_numeric(df[stat].astype(str).str.replace(',', ''), errors='coerce')
# Danh sách đội
teams = df['Team'].dropna().unique()
```

- Sử dụng thư viện matplotlib.pyplot để vẽ

```
# Vẽ và lưu histogram
for stat in plot_stats:
    # Histogram cho toàn giải
    vals_all = df[stat].dropna()
    plt.figure(figsize=(6,4))
    plt.hist(vals_all, bins='auto', edgecolor='black')
    plt.title(f'{stat}  All Players')
    plt.xlabel(stat)
    plt.ylabel('Count')
    plt.tight_layout()
    # Lưu file
    fname = plots_dir / f"{stat}_all.png"
    plt.savefig(fname)
    plt.close()
    # Histogram cho từng đội
    for team in teams:
        vals_team = df.loc[df['Team']==team, stat].dropna()
        if vals_team.empty:
            continue
        plt.figure(figsize=(6,4))
        plt.hist(vals_all, bins='auto', edgecolor='black')
        plt.title(f'{stat}  {team}')
        plt.xlabel(stat)
        plt.ylabel('Count')
        plt.tight_layout()
        fname = plots_dir / f"{stat}_{team.replace(' ', '_')}.png"
        plt.savefig(fname)
        plt.close()
```

4. (BAI2_4.py) : Dự đoán đội vô địch

- Đọc dữ liệu và chuẩn hóa dữ liệu các giá trị N/a thành giá trị nan trong numpy
- Lưu các cột để tìm chỉ số max theo mean(ngoài ra có thể làm hướng tìm max theo sum)

```
import pandas as pd
import numpy as np
from pathlib import Path

#  Đọc và chuẩn hóa dữ liệu
df = pd.read_csv( r'C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv', encoding='utf-8-sig', thousands=',')
df.replace('N/a', np.nan, inplace=True)
#  Xác định các feature và ép kiểu numeric
exclude = ['Name', 'Nation', 'Position', 'Team']
stats   = [c for c in df.columns if c not in exclude]
for stat in stats:
    df[stat] = pd.to_numeric( df[stat].astype(str).str.replace(',', ''), errors='coerce')
```

- Tìm mean từng đội mà ghi kết quả ra out.csv

```
# Tính mean theo team
team_means   = df.groupby('Team')[stats].mean()
best_by_mean = team_means.idxmax()
best_values  = team_means.max()

# Chuẩn bị report_df
report_df = pd.DataFrame({
    'Feature':    best_by_mean.index,
    'Top Team':   best_by_mean.values,
    'Mean Value': best_values.values.round(2)
})

# Chuẩn bị summary_df
lead_counts  = best_by_mean.value_counts().reset_index()
summary_df   = lead_counts.copy()
summary_df.columns = ['Team', 'Num Features highest score']

# Ghi ra out.csv
out_path = Path(__file__).parent / 'out.csv'
with open(out_path, 'w', encoding='utf-8-sig', newline='') as f:
    report_df.to_csv(f, index=False)
    f.write('\n')
    summary_df.to_csv(f, index=False)
```

- File out.csv như hình dưới đây:

Feature	Top Team	Mean Value
Age	Fulham	28.91
Matches_played	Liverpool	25.19
Starts	Brentford	18.33
Minutes	Liverpool	1643.1
Goals	Liverpool	3.81
Assists	Liverpool	2.86
Yellow Cards	Bournemouth	3.87
Red Cards	Arsenal	0.23
Expected_xG	Liverpool	3.68
Expected_xAG	Liverpool	2.68
PrgC	Manchester City	41.64
PrgP	Liverpool	83.95
PrgR	Liverpool	83.05
Per90_Gls	Manchester City	0.18
Per90_Ast	Liverpool	0.14
Per90_xG90	Aston Villa	0.19
Per90_xAG	Chelsea	0.15
GA90	Manchester Utd	2.65
Save%	Bournemouth	80
CS%	Aston Villa	58.8
PK_Save%	Everton	100
SoT%	Nott'ham Forest	39.04
SoT/90	Bournemouth	0.54
G/sh	Arsenal	0.14
Dist	Nott'ham Forest	19.12
Cmp	Liverpool	807.14
Total_Cmp%	Manchester City	86.5
TotDist	Liverpool	13712.95
Short_Cmp%	Manchester City	92.13
Medium_Cmp%	Manchester City	89.52
Long_Cmp%	Liverpool	60.28
KP	Liverpool	22.43
Expected_1/3	Liverpool	69.57
PPA	Liverpool	19.05
CrsPA	Fulham	4.5
Expected_PrgP	Liverpool	83.95
SCA	Liverpool	50.81

	SCA90	Liverpool	2.62
	GCA	Liverpool	6.52
	GCA90	Liverpool	0.33
	Tkl	Crystal Palace	33.14
	TklW	Crystal Palace	19.62
	Challenges_Att	Liverpool	28.95
	Challenges_Lost	Crystal Palace	14.14
	Blocks	Brentford	21.19
	Sh	Brentford	8.71
	Pass	Crystal Palace	14.9
	Int	Bournemouth	14.39
	Touches	Liverpool	1139.43
	Def Pen	Brentford	145.57
	Def 3rd	Brentford	366.48
	Mid 3rd	Liverpool	513.48
	Att 3rd	Manchester City	353.4
	Att Pen	Liverpool	57.38
	Take-Ons_Att	Arsenal	30.59
	Succ%	Liverpool	54.8
	Tkld%	Leicester City	48.3
	Carries	Manchester City	664.68
	ProDist	Manchester City	2051.96
	Carries_PrgC	Manchester City	41.64
	1/3	Manchester City	31.16
	CPA	Manchester City	14.48
	Mis	Nott'ham Forest	23.68
	Dis	Newcastle Utd	17.96
	Rec	Liverpool	798.24
	Receiving_PrgR	Liverpool	83.05
	Fls	Bournemouth	20.52
	Fld	Newcastle Utd	18.22
	Off	Nott'ham Forest	3.77
	Crs	Fulham	37.91
	Recov	Bournemouth	73.22
	Won	Brentford	27.52
	Lost	Crystal Palace	28.19
	Won%	Brentford	54.89

	Team	Num Features highest score
	Liverpool	27
	Manchester City	11
	Brentford	7
	Bournemouth	6
	Crystal Palace	5
	Nott'ham Forest	4
	Arsenal	3
	Fulham	3
	Aston Villa	2
	Newcastle Utd	2
	Everton	1
	Manchester Utd	1
	Chelsea	1
	Leicester City	1

- Thống kê cho thấy Liverpool là đội có số lần được highest scores nhất nên ta có thể dự đoán liverpool vô địch năm nay(thực tế đã vô địch)

III. (BAI3.py) : K-mean và PCA

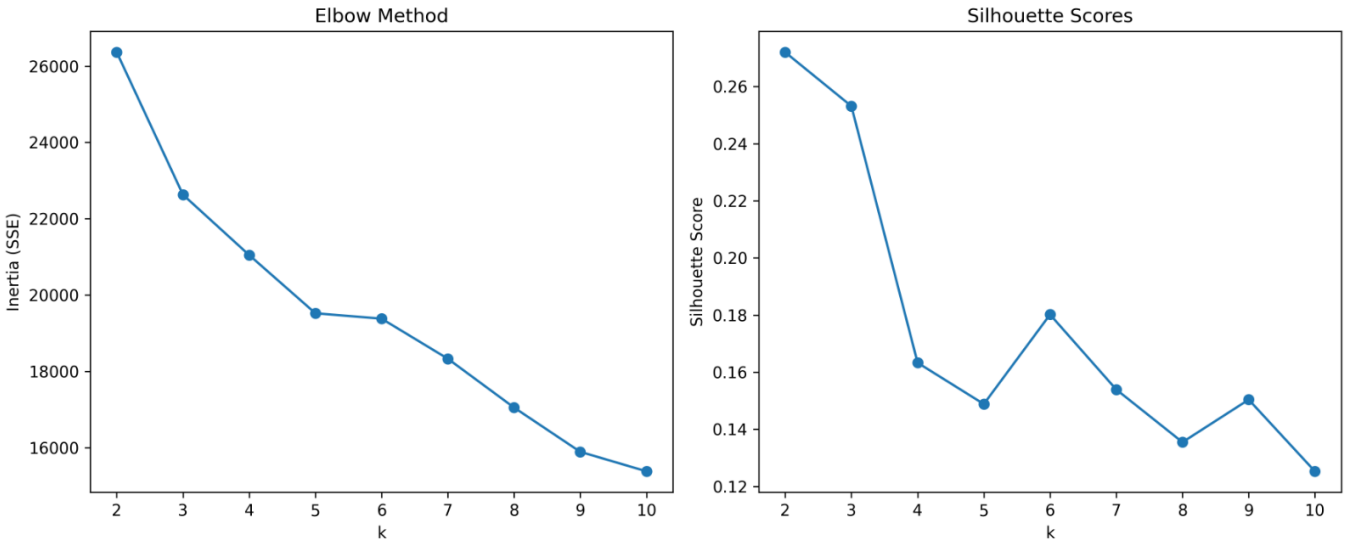
- Import thư viện cần dùng
- Đọc và chuẩn hóa dữ liệu. Fill các giá trị N/a thành mean và chuẩn hóa các thông số bằng StandardScaler

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Đọc dữ liệu và chuẩn hóa
df = pd.read_csv( r'C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv', encoding='utf-8-sig', thousands=',')
df_clean = df.drop(['Name', 'Team', 'Nation', 'Position'], axis=1)
df_clean = df_clean.apply(pd.to_numeric, errors='coerce')
means = df_clean.mean(numeric_only=True)
df_clean.fillna(means, inplace=True)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_clean)
```

- Để xác định K trong bài toán K-mean, chúng ta xử dụng 2 metric là đường “khuỷu tay” (Elbow) và điểm Silhouette scores rồi plot ra xem đâu là giá trị K tối ưu nhất

```
# Xác định k, sử dụng elbow (inertia) và silhouette
Ks = range(2, 11)
elbow = []
silhouettes = []

for k in Ks:
    km = KMeans(n_clusters=k, random_state=42)
    labels = km.fit_predict(X_scaled)
    elbow.append(km.inertia_)
    silhouettes.append(silhouette_score(X_scaled, labels))
```

- Dưới đây là kết quả thu được :

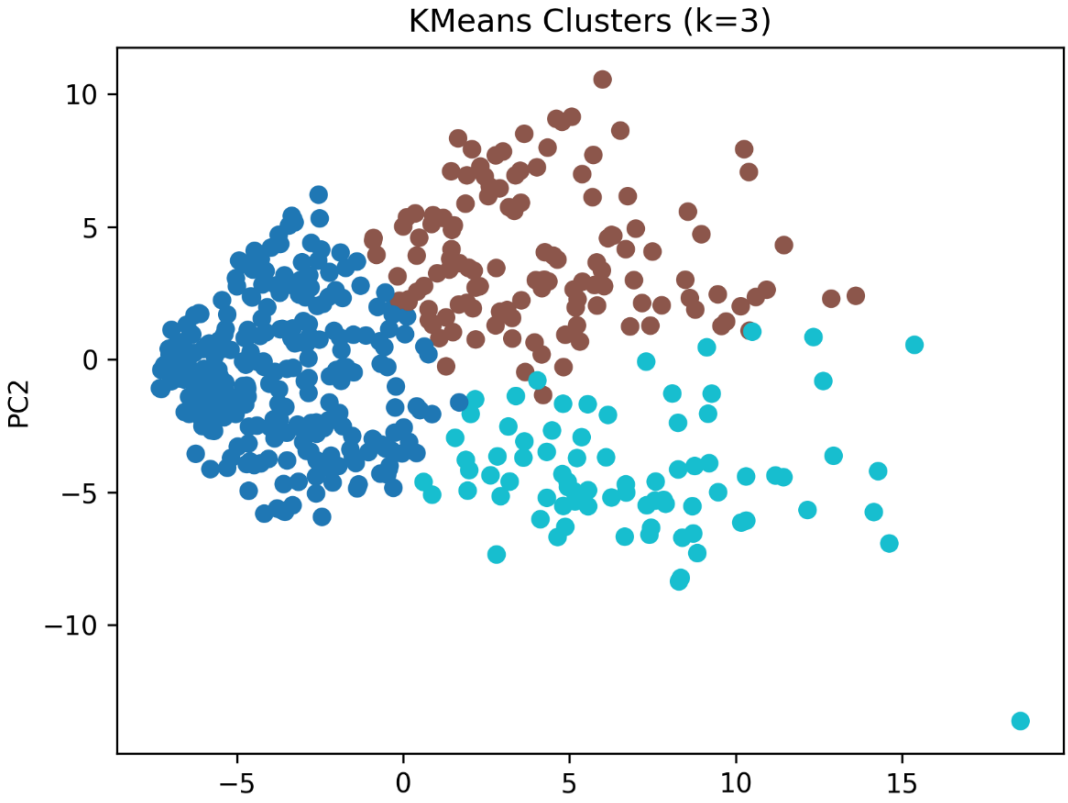


- Dựa vào hình ảnh thu được:
  - ✓ Với metrics Elbow, kết quả cho ta thấy k = 3, 4, 5 là đường khuỷu tay rõ nhất
  - ✓ Với metrics Silhouette, k = 2, 3 đạt điểm cao nhất

=> Chọn k = 3

- Sử dụng thư viện sklearn.cluster để lấy mode K-mean, sklearn.decomposition để giảm chiều dữ liệu (PCA) và matplotlib.pyplot để trực quan hóa dữ liệu

```
best_k = 3
# KMeans với k = 3 and và visualize  PCA
kmeans_opt = KMeans(n_clusters = best_k, random_state=42)
labels_opt = kmeans_opt.fit_predict(X_scaled)
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)
plt.figure()
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels_opt, cmap='tab10')
plt.title(f'KMeans Clusters (k={best_k})')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.savefig('cluster_pca.png', dpi=300)
plt.show()
```


KMeans Clusters (k=3)
PC2
10
5
0
−5
−10
−5
0
5
10
15

IV. Bài 4 : Chọn model và feature

1. (BAI4_1.py) : Thu thập dữ liệu giá trị chuyển nhượng của cầu thủ EPL chơi trên 900 phút:

- Truy cập trang web https://www.footballtransfers.com/en/values/players/most-valuable-players/playing-in-uk-premier-league
- Cách thu thập dữ liệu gần như giống với bài 1, để thuận lợi cho việc lấy dữ liệu và train model thì ngoài giá trị các cầu thủ, ta cũng lưu luôn tất cả các feature của các cầu thủ từ bài 1, ta coi như việc lấy dữ liệu giá trị chuyển nhượng là bước gán nhãn thủ công (manual labeling).
- Do 2 web có một chút lệch nhau về số lượng các cầu thủ, ta chỉ lưu những cầu thủ đã được lưu ở bài 1(do một số cầu thủ thiếu nếu lưu thì sẽ không có feature để train). Một số cầu thủ có ở bài 1 nhưng không có ở bài 4 cũng loại bỏ.
- Sử dụng webdriver trong thư viện selenium để truy cập trang web

```
driver = webdriver.Chrome()
driver.get("https://www.footballtransfers.com/en/values/players/most-valuable-players/playing-in-uk-premier-league")
```

- Tương tự bài 1, ta cũng dùng cấu trúc dữ liệu dict để lưu các thông tin cầu thủ

```
def normalize_name(name):
    nfkd = unicodedata.normalize('NFKD', name)
    return ''.join(c for c in nfkd if not unicodedata.combining(c)).lower().strip()

input_csv = r"C:\Users\Admin\OneDrive\Desktop\BTL PYTHON\BÀI 1\results.csv"
with open(input_csv, "r", encoding="utf-8-sig") as f:
    reader = csv.DictReader(f)
    data = list(reader)

Name = { normalize_name(r["Name"]): r for r in data }
```

- Dữ liệu của các cầu thủ gồm khoảng 22 page. Ta sẽ for lần lượt 22 page và dùng thư viện BeautifulSoup để parse HTML tĩnh, trỏ vào các thẻ chứa thông tin cần lấy. Sau khi xử lý xong trang hiện tại, tìm và click nút “Next” (selector tương ứng) để chuyển sang trang kế tiếp.

```
for page in range(1, 23):
    soup = BeautifulSoup(driver.page_source, "html.parser")
    table = soup.find("tbody", id="player-table-body")
    for tr in table.find_all("tr"):
        a = tr.find("a")
        tag = tr.find("span", class_="player-tag")
        if not a or not tag:
            continue
        name = a.get_text(strip=True)
        m = re.search(r"([\d\.]+)", tag.get_text())
        if not m: continue
        val = m.group(1)
        norm = normalize_name(name)
        if norm in Name:
            Name[norm]["ETV(€M)"] = val
    #next page
    if page < 22:
        btn = WebDriverWait(driver, 5).until(
            EC.element_to_be_clickable((By.CLASS_NAME, "pagination_next_button"))
        )
        btn.click()
        time.sleep(2)
```

- Cuối cùng, ta lưu cầu thủ + các feature + giá trị chuyển nhượng vào file data.csv(chỉ lọc các cầu thủ > 900 phút). Cột chuyển nhượng đặt tên là ETV(€M)

```
output_csv = "data.csv"
fields = list(data[0].keys())
if "ETV(€M)" not in fields:
    fields.append("ETV(€M)")

with open(output_csv, "w", newline="", encoding="utf-8-sig") as f:
    writer = csv.DictWriter(f, fieldnames=fields, restval="N/a")
    writer.writeheader()
    for r in data:
        etv = r.get("ETV(€M)", "").strip()
        mins = r.get("Minutes", "").replace(",", "").strip()
        if etv and mins.isdigit() and int(mins) > 900:
            writer.writerow(r)
```

2. (BAI4_2.py) : Lựa chọn các feature và model để dự đoán giá trị chuyển nhượng của cầu thủ

- Nhận xét : đây là bài toán hồi quy
- In ra các hệ số tương quan cho thấy các feature cao nhất ~ 0.48 + tính chất nhiều biến – ít mẫu (300 mẫu và 77 biến đặc trưng) đòi hỏi mô hình vừa có khả năng học quan hệ phi tuyến, vừa không quá dễ over-fit.

```
Hệ số tương quan (theo abs) so với target 'ETV(€M)':
SCA                 : 0.4831
Age                 : -0.4813
GCA                 : 0.4659
Att Pen             : 0.4598
Att 3rd             : 0.4559
Expected_xG         : 0.4464
KP                  : 0.4321
1/3                 : 0.4246
SCA90               : 0.4185
Per90_xG90          : 0.4143
PPA                 : 0.4134
Goals               : 0.4116
SoT/90              : 0.4087
Expected_xAG        : 0.4086
GCA90               : 0.4037
Expected_PrgP       : 0.4037
PrgP                : 0.4037
Carries_PrgC        : 0.3995
PrgC                : 0.3995
Dis                 : 0.3905
Per90_Gls           : 0.3835
Assists             : 0.3818
Receiving_PrgR      : 0.3743
PrgR                : 0.3743
Take-Ons_Att        : 0.3654
Per90_xAG           : 0.3626
CPA                 : 0.3585
Per90_Ast           : 0.3474
Mis                 : 0.3325
ProDist             : 0.3302
Fld                 : 0.3235
Rec                 : 0.3002
Carries             : 0.2810
Won%                : -0.2402
Expected_1/3        : 0.2332
Pass                : 0.2300
Mid 3rd             : 0.2251
Starts              : 0.2247
```

- Lựa chọn mô hình : HistGradientBoostingRegressor - Là một thuật toán **Gradient Boosting** (GB), tức xây dựng dàn cây quyết định (decision trees) theo kiểu **đua tiếp sức (gradient boosting)**: mỗi cây mới học phần còn thiếu (residual/gradient) của tổng dự đoán các cây trước đó, tận dụng sức mạnh của cây để học phi-tuyến, và tối ưu hoá tốc độ thông qua histogram binning.
- Ưu điểm so với bộ dữ liệu :

✓ HistGB có cơ chế regularization (learning_rate nhỏ, max_depth, min_samples_leaf) giúp tránh over-fit khi mẫu ít mà số biến nhiều.

✓ Histogram binning tự động gom giá trị feature, giảm noise của những biến ít thông tin.

✓ Mỗi feature chỉ được chia thành một số bin cố định (128 hay 256…) ⇒ giảm đáng kể phép so sánh tìm split so với GB truyền thống hoặc XGBoost/LightGBM khi số mẫu ~thấp–vừa.

✓ Ít tốn bộ nhớ, chạy nhanh ngay trên CPU phổ thông.

- Cài đặt :
- Import thư viện

```
import pandas as pd
from sklearn.model_selection import RepeatedKFold, GridSearchCV
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.inspection import permutation_importance
from sklearn.metrics import make_scorer, r2_score
```

- Tiền xử lý

✓ Đọc file dữ liệu 'data.csv', chỉ giữ lại các cột số và fill các giá giá N/a thành mean của cột đó

✓ HistGBR là một dàn cây quyết định chỉ quan tâm "so sánh" lớn hơn/nhỏ hơn một ngưỡng (split threshold), chứ không tính khoảng cách hay độ lớn tuyệt đối của giá trị nên có thể bỏ qua bước Scaler.

```
# 1. Đọc file
df = pd.read_csv('data.csv', encoding='utf-8-sig', na_values=['N/a'], thousands=',')
df = df.drop(['Team','Nation','Position','Name'], axis=1, errors='ignore')
for c in df.select_dtypes(include='object').columns:
    df[c] = pd.to_numeric(df[c].str.replace(',', ''), errors='coerce')
df.fillna(df.mean(), inplace=True)
```

- Định nghĩa target(cột cần dự đoán) là cột cuối

```
# 2. X : data, y : target
target = df.columns[-1]
X = df.drop(columns=[target])
y = df[target]
```

- Ta sẽ train thử qua để tìm top 10 các feature quan trọng(trong quá trình làm bài, với 300 cầu thủ thì lấy ~10-15 feature sẽ thu được kết quả tốt nhất)

```
# 3. Tìm top10 các feature quan trọng
base = HistGradientBoostingRegressor(random_state=42)
base.fit(X, y)
imp = permutation_importance(base, X, y, scoring='r2', n_repeats=15, random_state=42, n_jobs=-1)
imp_ser = pd.Series(imp.importances_mean, index=X.columns).sort_values(ascending=False)
top10 = imp_ser.iloc[:10].index.tolist()
```

- Dùng GridSearchCV trong sklearn.model_selection để tìm bộ tham số hợp lý
- Model được lấy sẵn trong thư viện sklearn.ensemble

```
# 4. Dữ liệu tham số
param_grid = {
    'max_iter': [100, 150, 200],
    'learning_rate': [0.03, 0.05],
    'max_depth': [2, 3, 4],
    'min_samples_leaf': [10, 15, 20],
    'max_bins': [128, 256]
}

hgb = HistGradientBoostingRegressor(random_state=42)

# 5. GridSearchCV
rkf = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42)
search = GridSearchCV(
    estimator=hgb,
    param_grid=param_grid,
    scoring=make_scorer(r2_score),
    cv=rkf,
    n_jobs=-1,
    verbose=2
)
```

- Fit mô hình và in ra kết quả của quá trình tìm kiếm tham số (GridSearchCV), mean cross-validated R^2

```
# 6. Fit
search.fit(X[top10], y)

# 7. Tham số tốt nhất
print("Best parameters found:")
print(search.best_params_)
print(f"Best CV R²: {search.best_score_:.4f}")

best_hgb = search.best_estimator_
best_hgb.fit(X[top10], y)
print("Final model trained on all data.")
```

- Kết quả thu được:

```
Best feature
Age Att Pen Att 3rd Total_Cmp% GCA Expected_1/3 Carries Per90_xG90 Tkld% PrgP
Best parameters found:
{'learning_rate': 0.05, 'max_bins': 128, 'max_depth': 3, 'max_iter': 150, 'min_samples_leaf': 15}
Best CV R²: 0.5226
```

Best Hyperparameters and CV R^2

Parameter	Value
learning_rate	0.05
max_bins	128
max_depth	3
max_iter	150
min_samples_leaf	15
CV R^2	0.5226

- Với bộ tham số phù hợp

✓ 'learning_rate': 0.05

✓ 'max_bins': 128

✓ 'max_depth': 3

✓ 'max_iter': 150

✓ 'min_samples_leaf': 15

- Đánh giá bằng Repeated 5×3 K-Fold CV, mô hình đạt CV R^2 = 0.5226, tức giải thích được khoảng 52.26% phương sai của biến mục tiêu.
- (Với chỉ ~300 bản ghi, mỗi lần chia fold chỉ có ~240 mẫu để train và ~60 mẫu để test. Dữ liệu nhỏ làm cho model khó "bắt" hết pattern, đồng thời dễ bị dao động khi gặp outlier hoặc vùng dữ liệu hiếm, nhiễu. Em đã thử một số model khác tuy nhiên R^2 chỉ dao động trong [0.2, 0.5]).