HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP

MÔN HỌC: LẬP TRÌNH PYTHON

Giảng viên hướng dẫn	**: Thầy Kim Ngọc Bách**
Họ và tên sinh viên	**: Đinh Thành Công**
Mã sinh viên	**: B23DCCE013**
Lớp	**: D23CQCE04-B**
Nhóm	**: 01**

Hà Nội – 2024

Contents

Preface

In an era of rapidly advancing data and automation, programming has become an essential skill not only in the field of information technology but also across various domains such as economics, engineering, science, and management. Among the numerous programming languages available today, Python stands out as a powerful, easy-to-learn, and versatile tool, widely used in data analysis, artificial intelligence, automation, web development, and many other applications. With its simple syntax, rich libraries, and a vast user community, Python has become the top choice for both beginners and professional programmers.

In the context of modern football, data analysis has emerged as a critical tool for evaluating player performance, developing team strategies, and making transfer decisions. This report aims to collect, process, and analyze statistical data of players competing in the English Premier League for the 2024-2025 season, based on data sourced from FBref.com and footballtransfers.com.

The report consists of four main sections:

1. Data Collection and Processing: Outlining the process of extracting, cleaning, and merging data from specialized statistical tables to create a comprehensive dataset in CSV format.
2. Descriptive Statistical Analysis: Ranking players, calculating team-based statistical metrics, visualizing data, and evaluating performance efficiency.
3. Player Clustering: Applying KMeans and PCA algorithms to group players based on statistical characteristics, aiding in similarity analysis.
4. Transfer Value Estimation: Building machine learning models to predict player market values, analyzing key features, and evaluating the performance of regression models.

Although this report has been completed with the utmost care, it is not without shortcomings. I sincerely welcome all feedback and evaluations from you, my teacher.

Thank you very much!

I:

1. Objectives

Outline the process of collecting, processing, and merging statistical data of players competing in the Premier League for the 2024–2025 season, provided by the website FBref.com. The output data is stored in CSV format, serving further in-depth analysis such as performance evaluation, player clustering, or building models to predict transfer values.

2. Tools and Libraries Used

Selenium: Automates browser operations to load web pages, especially for tables hidden within HTML comments.

BeautifulSoup: Extracts data from the retrieved HTML code.

Pandas: Processes and consolidates data into tabular format.

3. Data Sources

The data is sourced from various specialized statistical tables on the FBref website. These tables include:

Type of Statist	URL Link
standard	https://fbref.com/en/comps/9/stats/Premier-League-Stats
goalkeeping	https://fbref.com/en/comps/9/keepers/Premier-League-Stats
shooting	https://fbref.com/en/comps/9/shooting/Premier-League-Stats
passing	https://fbref.com/en/comps/9/passing/Premier-League-Stats
GCA	https://fbref.com/en/comps/9/gca/Premier-League-Stats
defense	https://fbref.com/en/comps/9/defense/Premier-League-Stats
possession	https://fbref.com/en/comps/9/possession/Premier-League-Stats
miscellaneou	https://fbref.com/en/comps/9/misc/Premier-League-Stats

4. Implementation Process

4.1. Browser Initialization and Page Loading

- Import the necessary libraries and create configuration options.
- Use Selenium in headless mode to open each URL.

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from bs4 import BeautifulSoup,Comment
import pandas as pd

options = Options()
options.add_argument('--headless')
options.add_argument('--disable-gpu')
options.add_argument('--no-sandbox')
```

Extract the page_source and convert it into a BeautifulSoup object.
Each statistical table is stored in a soup variable with a corresponding key.

```
soups = {}

for key, url in urls.items():
    driver = webdriver.Chrome(options=options)
    driver.get(url)
    html = driver.page_source
    soups[key] = BeautifulSoup(html, 'html.parser')
    driver.quit()
```

4.2. Data Table Extraction

The extract_table() function performs the following steps:

- Locate the table based on its ID.
- If the table is not found (due to being hidden in HTML comments), iterate through the comment sections and check each segment to find the table containing <tbody>.
- Extract row data (<tr>) and data cells (<td>).
- Column names are retrieved from the data-stat attribute of the first row.

```
def extract_table(soup, table_id):
    table = soup.find('table', id=table_id)
    if not table:
        comments = soup.find_all(string=lambda text: isinstance(text, Comment))
        for comment in comments:
            if table_id in comment:
                comment_soup = BeautifulSoup(comment, 'html.parser')
                table = comment_soup.find('table', id=table_id)
                break
    rows = table.find('tbody').find_all('tr')
    data_rows = [[td.text.strip() for td in row.find_all('td')]
                 for row in rows if not(row.get('class') and ('thead' in row.get('class')))]
    columns = [col['data-stat'] for col in rows[0].find_all('td')]
    return pd.DataFrame(data_rows, columns=columns)
```

4.3. Data Preprocessing

- The players_df is extracted from the "standard" table, including all players.
- Convert the "minutes" column to numeric type and remove unnecessary columns (e.g., number of matches, date of birth, composite metrics, etc.).
- Retain only players with more than 90 minutes of playtime to ensure statistically robust data.

```
players_df = extract_table(soups['standard'],table_id="stats_standard")
keepers_df = extract_table(soups['goalkeeping'], table_id="stats_keeper")
shooting_df = extract_table(soups['shooting'],table_id="stats_shooting")
passing_df = extract_table(soups['passing'],table_id="stats_passing")
gca_df = extract_table(soups['gca'],table_id="stats_gca")
defense_df = extract_table(soups['defense'],table_id="stats_defense")
possession_df = extract_table(soups['possession'],table_id="stats_possession")
misc_df = extract_table(soups['misc'],table_id="stats_misc")

players_df['nationality'] = players_df['nationality'].str.split().str[-1]
players_df['minutes'] = players_df['minutes'].str.replace(',','')
players_df['minutes'] = pd.to_numeric(players_df['minutes'], errors='coerce')
players_df = players_df[players_df['minutes'] > 90]
```

4.4. Data Table Processing

- From each sub-table (keepers, shooting, passing, etc.), retain only the important and relevant statistical columns.
- The retained columns include:

  o Na

  tion o

  Team

  o Po

  sition o

  Age

  o Pl

  aying

  Time:

  matches

  played,

  starts,

  minutes o

  Performan

  ce: goals,

assists,
yellow
cards, red
cards o
Expected:
expected
goals (xG),
expedted
Assist
Goals
(xAG) o
Progressio
n: PrgC,
PrgP, PrgR
o Per 90
minutes:
Gls, Ast,
xG, xGA o
Goalkeepi
ng:

- ▪ Performance: goals against per 90mins (GA90), Save%, CS%
- ▪ Penalty Kicks: penalty kicks Save%

o Shooting:

- ▪ Standard: shoots on target percentage (SoT%), Shoot on Target per 90min (SoT/90), goals/shot (G/sh), average shoot distance (Dist)

o Passing:

- ▪ Total: passes completed (Cmp),Pass completion (Cmp%), progressive passing distance (TotDist)
- ▪ Short: Pass completion (Cmp%),
- ▪ Medium: Pass completion (Cmp%),
- ▪ Long: Pass completion (Cmp%),
- ▪ Expected: key passes (KP), pass into final third (1/3), pass into penalty area (PPA), CrsPA, PrgP

o Goal and Shot Creation:

- SCA: SCA, SCA90
- GCA: GCA, GCA90

o Defensive Actions:

- Tackles: Tkl, TklW
- Challenges: Att, Lost
- Blocks: Blocks, Sh, Pass, Int

o Possession:

- Touches: Touches, Def Pen, Def 3rd, Mid 3rd, Att 3rd, Att Pen
- Take-Ons: Att, Succ%, Tkld%
- Carries: Carries, ProDist, ProgC, 1/3, CPA, Mis, Dis
- Receiving: Rec, PrgR

o Miscellaneous Stats:

- Performance: Fls, Fld, Off, Crs, Recov
- Aerial Duels: Won, Lost, Won%

```
players_df.drop(columns=[
    'birth_year','minutes_90s','goals_assists','goals_pens','pens_made','pens_att',
    'npxg','npxg_xg_assist','goals_assists_per90','goals_pens_per90',
    'goals_assists_pens_per90','xg_xg_assist_per90','npxg_per90',
    'npxg_xg_assist_per90','matches'
], errors='ignore', inplace=True)
keepers_df = keepers_df[['player','team','gk_goals_against_per90','gk_save_pct','gk_clean_sheets_pct','gk_pens_save_pct']]
shooting_df = shooting_df[['player','team', 'shots_on_target_pct', 'shots_on_target_per90', 'goals_per_shot', 'average_shot_distance']]
passing_df = passing_df[['player','team', '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_df = gca_df[['player','team','sca','sca_per90','gca','gca_per90']]
defense_df = defense_df[['player','team','tackles','tackles_won','challenges','challenges_lost','blocks','blocked_shots'
                        ,'blocked_passes','interceptions']]
possession_df = possession_df[['player','team','touches','touches_def_pen_area','touches_def_3rd','touches_mid_3rd',
                   'touches_att_3rd','touches_att_pen_area','take_ons','take_ons_won_pct','take_ons_tackled_pct','carries'
                   ,'carries_progressive_distance','progressive_carries','carries_into_final_third','carries_into_penalty_area',
                   'miscontrols','dispossessed','passes_received','progressive_passes_received']]
misc_df = misc_df[['player','team','fouls','fouled','offsides','crosses','ball_recoveries','aerials_won'
                  ,'aerials_lost','aerials_won_pct']]
```

· Sequentially use merge() based on ['player', 'team'] to combine all tables into a single df_final table.

· Missing data is filled with 'N/a'.

· Sort the data by the player's first name.

· Save the data to a CSV file with UTF-8 encoding (with BOM) for optimal compatibility with Excel.

· File name: results.csv

· The output is a single comprehensive table containing multidimensional information about each player, stored in the file results.csv.

```
df_final = players_df.merge(keepers_df, how='left', on=['player','team']) \
                     .merge(shooting_df, how='left', on=['player','team']) \
                     .merge(passing_df, how='left', on=['player','team']) \
                     .merge(gca_df, how='left', on=['player','team'])\
                     .merge(defense_df,how='left',on=['player','team'])\
                     .merge(possession_df,how='left',on=['player','team'])\
                     .merge(misc_df,how='left',on=['player','team'])
df_final.fillna('N/a', inplace=True)
df_final_sorted = df_final.loc[df_final['player'].str.split().str[0].argsort()]
df_final_sorted.to_csv("results.csv", index=False, encoding='utf-8-sig')
print(df_final_sorted)
```

II:

After collecting and processing player data from various statistical tables on fbref.com, the next step is to conduct descriptive statistical analysis, rank players based on individual metrics, and visualize the data to gain a comprehensive understanding of the performance of players and teams in the Premier League for the 2024–2025 season.

1. Data Preparation

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import os
```

The dataset used is results.csv, which includes all players with over 90 minutes of playtime, along with standardized and merged technical metrics from various tables such as shooting, passing, defending, ball possession, goalkeeping, etc.

The initial data is further processed by:

- Replacing all "N/a" or NaN values with 0 during statistical analysis to avoid errors or biases.
- Converting numeric data columns from string format to float format to enable calculations.

```
df = pd.read_csv("results.csv")
teams = sorted(df['team'].unique())
col = [i for i in df.columns][5:]
```

2. Ranking Top 3 Players by Each Metric

To evaluate and compare individual performance, the system will identify:

- The top 3 players with the highest values for each data column.
- The top 3 players with the lowest corresponding values.

The results are exported to a file named top_3.txt, formatted clearly with headings for each metric, including the full player names and their respective values. This facilitates quick identification of players who are performing exceptionally or underperforming in the season.

```
with open('top_3.txt',"w", encoding='utf-8') as f:
    for i in col:
        tmp = []
        f.write("-"*10 + str(i).replace("_", " ").title() +"-"*10 + '\n')
        for index,row in df.iterrows():
            if row[i]!='N/a':
                tmp.append([f'{row['player']}',f'{row[i]}'])
        tmp.sort(key = lambda k:(float(k[1])))
        f.write("Top 3 highest:\n"+"\n")
        for j in range(-1,-4,-1):
            f.write((tmp[j][0]) +": " + tmp[j][1] + "\n" + "\n")
        f.write("Top 3 lowest:\n" +"\n")
        for j in range(3):
            f.write(tmp[j][0] + ": " +tmp[j][1] + "\n" + "\n")
        f.write("="*30 + "\n" + "\n")
```

The steps are performed as follows:

- Open a text file named top_3.txt for writing data. If the file does not exist, the program will create a new one. The data will be written with UTF-8 encoding, suitable for Vietnamese. The variable f represents this file.
- Iterate through each player statistical metric, such as goals, assists, or successful passes. The list col contains the names of these statistical columns.
- Initialize a temporary list tmp to store the player names and their corresponding statistical values for the current metric.
- Iterate through each row in the dataset, with each row corresponding to a player.
- Process only players with valid data for the metric being evaluated, excluding undefined values.
- If valid data exists, add a pair consisting of the player's name and their statistical value to the temporary list.
- Sort the tmp list in ascending order based on the statistical values.
- Iterate through the last three elements of the tmp list, as the list is sorted in ascending order, so the last three elements have the highest values.
- Write the names and statistical values of these players to the file.
- Iterate through the first three elements of the tmp list, as these are the players with the lowest values for the metric.
- Write their names and statistical values to the file.

The results are saved in the top_3.txt file as follows:

3.Descriptive Statistical Analysis by Metric and Team

In this step, the program performs descriptive statistical calculations, including:

- Mean
- Median
- Standard Deviation
  These values are calculated for all players in the league (combined) and for each team individually. This allows users to:
- Compare the level of consistency across teams.
- Identify teams with stable performance or those with standout players.
  The results are saved to a file named results2.csv, where each row corresponds to a team or the entire league, and each column represents a statistical measure for a technical metric.

The steps are performed as follows:

- Create a list to store statistical results for all players and for each team individually. A statistical row representing all players is marked with the label "all." A reference copy of the original dataset is created for separate processing.
- Iterate through each player statistical metric, such as shots or tackles:
  - The to_numeric function converts all values in the column to float format for calculations. If an error occurs, the program automatically converts the value to missing.
  - Fill all missing cells with zero to ensure data consistency during calculations.
  - Calculate general statistics for all players, including median, mean, and standard deviation. o The results are added to a dictionary row initialized earlier.

```
data = []
row = {
    '': 'all'
}
df2 = df
for i in col:
    df2[i] = pd.to_numeric(df2[i], errors='coerce')
df2.fillna(0,inplace = True)
for i in col:
    row[f"Median of {i}"] =  df2[i].median()
    row[f"Mean of {i}"] = df2[i].mean()
    row[f"Std of {i}"] = df2[i].std()
data.append(row)
```

· Iterate through each team present in the dataset.

· Filter players belonging to the team being evaluated and store them in a separate table.

· Calculate the median, mean, and standard deviation for the current metric for that team.

· After completing the statistical calculations for the team, add this row to the results list.

· Convert the list of statistical rows into a complete dataset and save it to the file results2.csv.

4. Plotting Data Distribution Charts

Data visualization is performed to support qualitative analysis:

- For each metric, the system generates a histogram showing the data distribution across all players in the league.
- Additionally, the system creates separate histograms for each team to analyze internal data dispersion.
- All charts are saved in .png format in two directories for easy reuse without needing to rerun the code:
    - image/all_player/ – contains charts for the entire league.
    - image/eachTeam/ – contains charts for individual teams.
      First, create a parent directory named image with two subdirectories, all_player and eachTeam, using a command that avoids errors if the directories already exist by including a parameter to keep them intact.

```
os.makedirs("image/all_player", exist_ok=True)
os.makedirs("image/eachTeam", exist_ok=True)
```

First Loop: Plotting Charts for All Players

- Iterate through each statistical metric for which a chart needs to be plotted.
- Inside the loop:
    - df_copy equals df3: This is not strictly necessary since operations are performed directly on df3.
    - tmp equals column x: Retrieve all data for the metric being evaluated.
    - Replace missing values in tmp with zero: Ensures no errors occur during chart plotting.
- Plot the chart:

- Use the histogram function to display the data distribution. ○ Set the title as the distribution of the current metric.
- Label the x-axis with the metric name. ○ Label the y-axis as the frequency of occurrence. ○ Save the chart to the all_player directory, with the file name based on the metric, replacing forward slashes with the word "per" to avoid file naming errors. ○ Close the chart to prepare for the next plotting iteration.

```
for x in col:
    df_copy = df3
    tmp = df_copy[x]
    tmp.fillna(0,inplace=True)
    plt.hist(tmp)
    plt.title(f"phân bố của chỉ số {x}")
    plt.xlabel(x)
    plt.ylabel("Frequency")
    plt.savefig(f'image/all_player/statistic_of_{x.replace('/', '_per_')}.png',dpi=300)
    plt.close()
```

Second Loop: Plotting Charts for Each Team

- for team in teams: Iterate through the list of teams present in the dataset. Inside the loop:
- tmp equals the data of the current team: Filter out players belonging to a specific team.
- for x in col: Iterate through each metric to be plotted. ○ df_copy equals column x of the current team.
    - Replace missing values with zero to avoid errors.

Plot the Chart

- Plot a histogram for the current metric of the team being evaluated.
- Set the chart title to include the metric name and the team name.
- Label the x-axis and y-axis similarly to the charts for all players.
- Save the image to the eachTeam directory with a file name that includes both the metric and the team name. Forward slashes in the metric name are replaced to avoid file naming errors.

```
for team in teams:
    tmp = df3[df3['team'] == team]
    for x in col:
        df_copy = tmp[x]
        df_copy.fillna(0,inplace = True)
        plt.hist(df_copy)
        plt.title(f"phân bố của chỉ số {x} đội {team}:")
        plt.xlabel(f'{x} of team {team}')
        plt.ylabel("Frequency")
        plt.savefig(f"image/eachTeam/statistic_of_{x.replace('/', 'per')}_team_{team}.png",dpi = 300)
        plt.close()
```

5. Identifying the Most Effective Team

The final step is to evaluate overall which team has the most players leading in individual metrics. The method is as follows:

- For each metric, identify the player with the highest value. Increment a counter for the team that player belongs to.

Steps to Perform:

- Create a dictionary named dict, where:
  - Each key is the name of a team.
  - The initial value is zero, representing the number of times the team has a player leading in a metric.
- Iterate through each row (corresponding to a player).
- Skip players with an "N/a" value for the metric being evaluated.
- If the current player's value is greater than or equal to the highest recorded value (m), update:
  - m to the new value. o t to the name of the player's team. If a team t has a player leading in the current metric, increment that team's counter by one in the dictionary.
- Convert the dictionary into a list of pairs (team, number of times leading).
- Sort in descending order based on the number of times a team's players lead in a metric.
- Print the results.

```
df4 = df
dict = {}
for team in teams: dict[team] = 0
for c in col:
    t, m = "", 0
    for index, row in df.iterrows():
        if row[c] != 'N/a':
            if float(row[c]) >= m:
                m = float(row[c])
                t = row['team']
    if t:
        dict[t] += 1
list = sorted(dict.items(), key=lambda x: (-x[1]))
for i in list:
    print(i[0],": ",i[1])
print("the Team, which is performing the best in the 2024-2025 Premier League season is",list[0][0])
```

The obtained result is:

```
Liverpool :  21
Manchester City :  6
West Ham :  5
Arsenal :  4
Aston Villa :  4
Brentford :  4
Everton :  4
Fulham :  4
Chelsea :  3
Leicester City :  3
Manchester Utd :  3
Nott'ham Forest :  3
Bournemouth :  2
Brighton :  2
Newcastle Utd :  2
Ipswich Town :  1
Tottenham :  1
Wolves :  1
Crystal Palace :  0
Southampton :  0
the Team, which is performing the best in the 2024-2025 Premier League season is Liverpool
```

III:

1. Objectives

The objective of this section is to analyze the similarity between football players by using the KMeans clustering method. From the player statistical data (including shooting, passing, defending, ball possession, etc.), we aim to group players into clusters with similar characteristics.

Since the data is high-dimensional (with many statistical columns), we also apply the PCA (Principal Component Analysis) technique to reduce the data to two dimensions for visualization purposes.

2. Data Preprocessing

```
df = pd.read_csv("results.csv")
teams = sorted(df['team'].unique())
columns = [i for i in df.columns][5:]
```

Handling Missing Values and Data Type Conversion

- Replace 'N/a' values with NaN.
- Fill missing values with 0.
- Convert all feature columns to the float data type.

```
df = df.replace('N/a', np.nan)
df.fillna(0,inplace = True)
df[columns] = df[columns].astype(float)
```

3. Data Standardization

- The data is standardized using StandardScaler to scale all features to the same range (mean of 0 and standard deviation of 1).
- Standardization is mandatory for KMeans to ensure that no single column dominates the clustering results due to having larger values.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[columns])
```

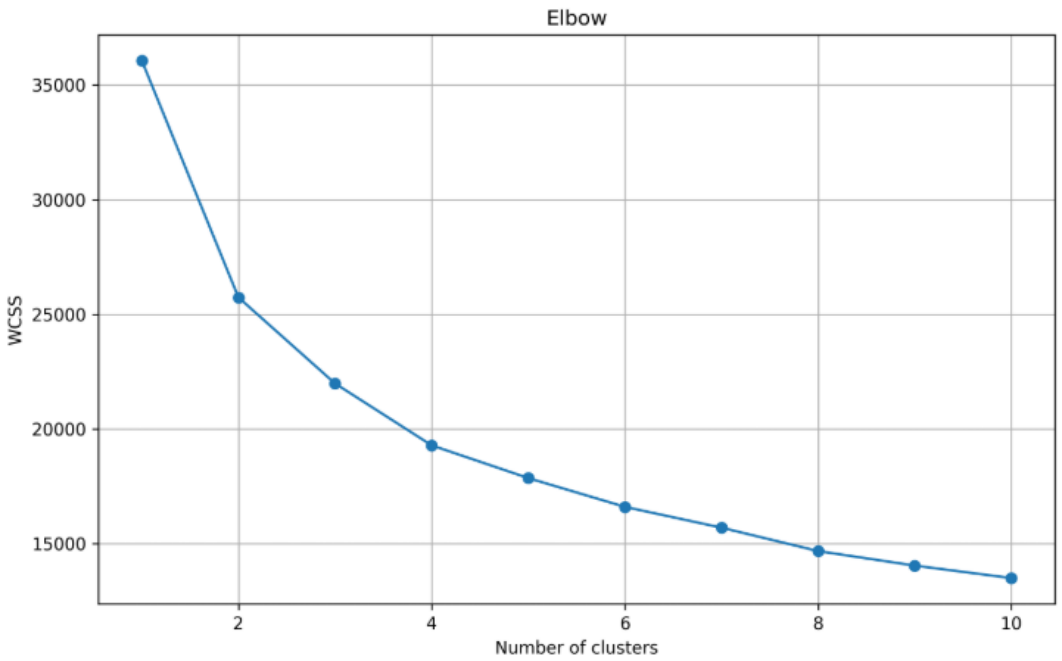
4. Determining the Optimal Number of Clusters Using the Elbow Method

- Apply the KMeans algorithm with the number of clusters ranging from 1 to 10.
- For each value of k, record the within-cluster sum of squares (WCSS).
- Plot a chart to identify the “elbow point,” where increasing the number of clusters no longer significantly reduces the WCSS.

```python
wcss = []
for k in range(1,11):
    kmean = KMeans(n_clusters=k, random_state=0)
    kmean.fit(X_scaled)
    wcss.append(kmean.inertia_)
plt.figure(figsize=(10,6))
plt.plot(range(1,11),wcss, marker='o')
plt.title('Elbow')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.savefig('image/bai3/ElbowMethod.png',dpi=300)
plt.show()
plt.close()
```

The chart image is saved as ElbowMethod.png in the image/bai3 directory.

5. Performing Clustering with KMeans



Based on the Elbow chart, we select the number of clusters as k = 4.

- Apply KMeans to cluster the entire dataset.
- The cluster centers are stored in centers.

6. Dimensionality Reduction with PCA

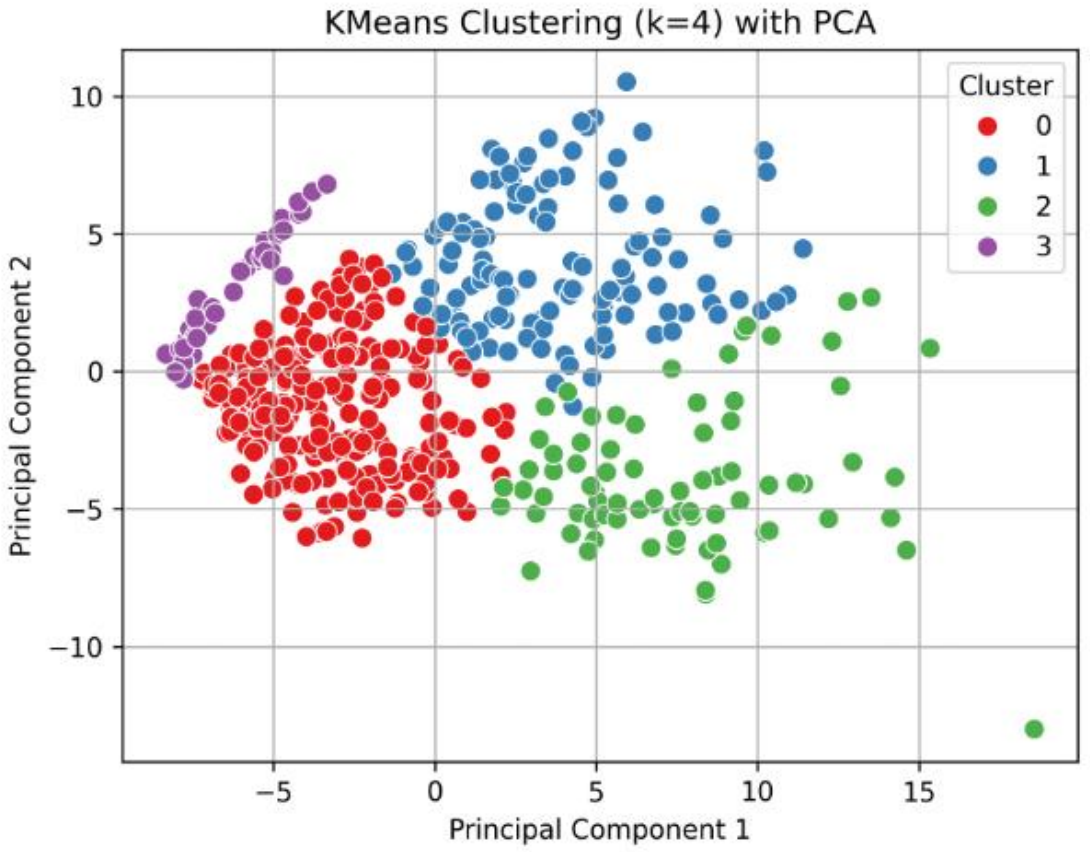
- PCA (Principal Component Analysis) is a dimensionality reduction method that retains as much of the original information as possible.
- Apply PCA to reduce the data to 2 dimensions for visualization purposes.
- The data after PCA is stored in a DataFrame, with cluster labels from KMeans added.

```
pca = PCA(n_components = 2)
pca.fit(X_scaled)
data_pca = pca.transform(X_scaled)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2'])
data_pca['Cluster'] = kmean.labels_
```

7. Visualization of Clustering Results

Draw a 2D scatter plot with the axes as PC1 and PC2.
Each point represents a player, and the color indicates the player's cluster.
The cluster centroids are shown using yellow star markers.
The plot is saved as PCA.png in the image/bai3 folder.

```
sns.scatterplot(data=data_pca, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=60)
plt.scatter(centers[:, 0], centers[:, 1], c='yellow', marker='*', s=300, edgecolors='black', label='Centroids')
plt.title(f'KMeans Clustering (k={k}) with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.legend(title='Cluster')
plt.savefig('image/bai3/PCA.png',dpi=300)
plt.show()
plt.close()
```



IV:

1. Problem Objective

The objective of this project is to estimate the transfer value (estimated market value) of players competing in the English Premier League (Premier League) for the 2024–2025 season, based on in-depth performance statistics from the season. Player value is a key factor in transfers, squad building, and financial performance analysis.

Main requirements:

Use only data from players who have played more than 900 minutes in the season (to exclude substitutes or rarely used players).

Collect player values from *footballtransfers.com*.

Build a machine learning model to estimate player value based on technical features.

Compare the performance of different regression models and select the best one.

Analyze the features that have the greatest impact on player value.

2. Data Collection

2.1. Player Value (from footballtransfers.com)

The website *footballtransfers.com* provides real-time estimated transfer values for players. The data is collected through POST requests to the site's API.

The *footballtransfers.com* website provides player value data via an internal API using the POST method at the following endpoint:

```
url = 'https://www.footballtransfers.com/en/values/actions/most-valuable-football-players/overview'
```

This API requires several parameters to be included in the POST body to filter the data, such as: league, page number, number of items per page, sort order, etc.

To access the API as a regular user, it is necessary to include HTTP headers that emulate a browser:

```
headers = {
    'user-agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
    'x-requested-with':'XMLHttpRequest',
    'referer': 'https://www.footballtransfers.com/en/values/players/most-valuable-players/playing-in-uk-
    'content-type': 'application/x-www-form-urlencoded; charset=UTF-8'
}
```

These headers help bypass the website's bot detection mechanisms and ensure successful queries.

Function to Collect Data from Each Page

The player data is divided into multiple pages (25 players per page). The get_data(i) function is written to automatically send requests to retrieve data from each page. Details:

- tournamentId = 31 is the internal code representing the Premier League.

- Other fields are left blank to avoid filtering and to retrieve all players in this league.
- The returned result is in JSON format, and the list of players is extracted via the "records" key.

```
rows = []
def get_data(i):
    payload = {
        'orderBy': 'estimated_value',
        'orderByDescending': 1,
        'page': i,
        'pages': 0,
        'pageItems': 25,
        'positionGroupId': all,
        'mainPositionId': all,
        'playerRoleId': all,
        'age': all,
        'countryId': all,
        'tournamentId': 31
    }
    x = requests.post(url,data=payload, headers=headers)
    data = x.json()
    players = data.get("records", [])
    for i in players:
        rows.append(i)
```

2.2. Data Merging and Filtering

After obtaining both the performance statistics and player values:

- Merge the two tables based on the "player" column (player name).
- Keep only players who have played 900 minutes or more.
- Remove records with missing estimated value data.
- The cleaned dataset is saved as:
  player_transfer_value.csv – containing player names, minutes played, and estimated value.

3. Data Preprocessing

Two datasets are used:

results.csv: contains player statistics on technical skills, passing, defense, assists, etc., collected from FBref.

player_transfer_value.csv: includes player names, minutes played, and estimated transfer value, collected from *footballtransfers.com*.

The data preprocessing steps include:

Merge the two tables based on player and minutes, keeping only players with over 900 minutes played to ensure statistical reliability.

Remove missing values (NaN) and replace "N/a" with np.nan.

Convert currency values (e.g., "€95M") to numerical (float) type using the convert_market_value function.

Drop unnecessary columns, keeping only numerical features for training.

```
def convert_market_value(value):
    if isinstance(value, str):
        value = value.replace('€', '').strip()
        if 'M' in value:
            return float(value.replace('M', '').strip()) * 1e6
        elif 'B' in value:
            return float(value.replace('B', '').strip()) * 1e9
        else:
            try:
                return float(value)
            except ValueError:
                return np.nan
    return value
```

```
stats_df = pd.read_csv('results.csv')
values_df = pd.read_csv('player_transfer_value.csv')

df_filtered = stats_df.merge(values_df,how = 'left', on=['player','minutes'])
df_filtered = df_filtered[df_filtered['minutes'] >= 900]
df_filtered.dropna(inplace=True)
df_filtered.replace('N/a', np.nan, inplace=True)
numerical_features = [col for col in df_filtered.columns][5:]

df_filtered['estimated_value'] = df_filtered['estimated_value'].apply(convert_market_value)
df_filtered.fillna(0, inplace=True)
```

4. Estimation Method and Model Selection

Objective: Build a model to predict a player's *estimated_value* based on technical features.

4.1. Models Used

The following models were evaluated:

LinearRegression

Ridge Regression

Lasso Regression
RandomForestRegressor

XGBRegressor (XGBoost)

Procedure:

Split the data into a training set (80%) and a test set (20%).

Standardize input data using StandardScaler.

Train the models and evaluate them using three metrics:

MAE (Mean Absolute Error)

RMSE (Root Mean Squared Error)

R^2 (R-squared – model goodness of fit)

Print a results table and visualize it with three charts (MAE, RMSE, and R^2) to compare the models.

	Model	MAE (€)	RMSE (€)	R^2
0	LinearRegression	21132935	27616829	-0.1318
1	Ridge	20540446	26470671	-0.0398
2	Lasso	21262273	27641051	-0.1338
3	RandomForest	14873018	19727164	0.4225
4	XGBoost	15780065	20910908	0.3511

All models are incorporated into a pipeline with StandardScaler and then evaluated using three metrics:

MAE (Mean Absolute Error)

RMSE (Root Mean Squared Error)

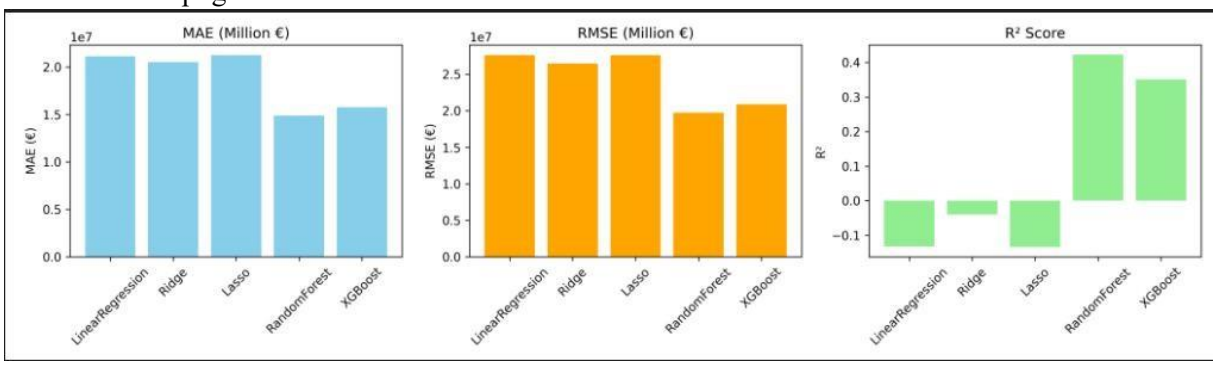
R^2 (Coefficient of Determination – indicates model goodness of fit)

4.2. Model Evaluation
After training and evaluation:

RandomForest and XGBoost delivered the best results in terms of MAE, RMSE, and R^2.

Linear Regression, Ridge, and Lasso showed average performance (due to the limitation of linear relationships).

The results are visualized using bar charts (MAE, RMSE, R^2) and saved as the image sosanhmohinh.png.



5. Selecting Important Features

After identifying Random Forest as the best model:

Use .feature_importances_ to determine the impact of each feature on the transfer value.

Select the top 15 most important features, including metrics related to scoring, chance creation, defense, and ball control.

Retrain the model using only these top 15 features → the results remain very good, simplifying the model and improving efficiency.

```
X = df_filtered[numerical_features].drop(columns=['estimated_value'])
y = df_filtered['estimated_value']
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X, y)
importances = model.feature_importances_
feature_names = X.columns

# Tạo DataFrame sắp xếp theo độ quan trọng
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

top_features = feature_importance_df['Feature'].head(15).tolist()
print("Top features:", top_features)
```

6. Prediction & Saving Results

Use the trained Random Forest model with the top 15 features to predict transfer values for all players.

Calculate the gap between actual and predicted values → helps identify undervalued or overvalued players.

Save the complete results including:

player

predicted_value

estimated_value

gap (€)

The results are saved as: player_value_predictions.csv

```python
# Huấn luyện lại mô hình với 15 đặc trưng
X_top15 = df_filtered[top_features]
y = df_filtered['estimated_value']
X_train, X_test, y_train, y_test = train_test_split(X_top15, y, test_size=0.2, random_state=0)

model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:,.0f} €")
print(f"RMSE: {rmse:,.0f} €")
print(f"R²: {r2:.4f}")

df_filtered['predicted_value'] = model.predict(df_filtered[top_features])
df_filtered['gap (€)'] = df_filtered['predicted_value'] - df_filtered['estimated_value']
df_filtered['abs_gap (€)'] = df_filtered['gap (€)'].abs()
df_filtered = df_filtered[['player','predicted_value','estimated_value','gap (€)']]
df_filtered.to_csv('player_value_predictions.csv', index=False, encoding='utf-8-sig')
```