I. Bài tập lớn 1:

```
import undetected_chromedriver as uc
from bs4 import BeautifulSoup as bs
import time
import pandas as pd
import numpy as np
```

➔ Khai báo các thư viện:

- undetected_chromedriver: chạy Chrome không bị phát hiện bởi các trang web chống bot.
- BeautifulSoup: phân tích và trích xuất dữ liệu từ mã HTML.
- time: để tạm dừng chương trình (delay khi chờ trang web tải).
- pandas, numpy: xử lý và phân tích dữ liệu bảng.

1. Lấy dữ liệu từ bảng standard:

```
url_players = "https://fbref.com/en/comps/9/stats/Premier-League-Stats"
```

➔ Truy cập vào URL standard stats

```
options1 = uc.ChromeOptions()
options1.add_argument("--no-sandbox")
options1.add_argument("--disable-dev-shm-usage")
options1.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options1, headless=False) as driver:
    driver.get(url_players)
    time.sleep(30)
    html_players = driver.page_source
```

➔ Mở trình duyệt, chờ 30 giây cho trang tải xong, lấy mã HTML.

```python
player_columns = {
    'player': 'Player',
    'nationality': 'Nationality',
    'team': 'Team',
    'position': 'Position',
    'age': 'Age',
    'games': 'MP',
    'games_starts': 'Starts',
    'minutes': 'Mins',
    'goals': 'Goal',
    'assists': 'Assists',
    'cards_yellow': 'Yellow cards',
    'cards_red': 'Red_cards',
    'xg': 'xG',
    'xg_assist': 'xGA',
    'progressive_carries': 'PrgC',
    'progressive_passes': 'PrgP',
    'progressive_passes_received': 'PrgR',
    'goals_per90': 'Gls',
    'assists_per90': 'Ast',
    'xg_per90': 'xG/90',
    'xg_assist_per90': 'xGA/90'
}
```

➔ Định nghĩa **các cột cần lấy** từ bảng HTML -> tên cột tương ứng trong DataFrame.

```python
soup = bs(html_players, 'html.parser')
div = soup.find('div', id="div_stats_standard")
table = div.find('table', id='stats_standard')
tbody = table.find('tbody')
rows = tbody.find_all('tr')
```

➔ Dùng BeautifulSoup để phân tích HTML -> tìm bảng stats_standard

```
players_data = []

for row in rows:
    player = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in player_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            if stat == 'nationality':
                parts = val.split()
                val = parts[-1] if parts else ''
            player[stat] = val
```

➔ Với mỗi hang, lấy theo từng cột data-stat

```
try:
    minutes = int(player.get('minutes', '0').replace(',', ''))
    if minutes >= 90:
        players_data.append(player)
except:
    continue
```

➔ Chỉ lưu các cầu thủ có trên 90p thi đấu.

```
df_players = pd.DataFrame(players_data)
```

➔ Tạo Data Frame để chứa các chỉ số

2. Lấy dữ liệu của thủ môn:

```
url_keepers = "https://fbref.com/en/comps/9/keepers/Premier-League-Stats"
```

➔ URL của các chỉ số thủ môn

```
options2 = uc.ChromeOptions()
options2.add_argument("--no-sandbox")
options2.add_argument("--disable-dev-shm-usage")
options2.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options2, headless=False) as driver:
    driver.get(url_keepers)
    time.sleep(10)
    html_keepers = driver.page_source

keeper_columns = {
    'player': 'Player',
    'gk_minutes': 'Gk_Mins',
    'gk_goals_against_per90': 'GA90',
    'gk_save_pct': 'Save%',
    'gk_clean_sheets_pct': 'CS%',
    'gk_pens_save_pct': 'Penalty Save%'
}
```

➔ Mở trình duyệt, chờ 10, lấy mã HTML và khai báo các chỉ số của thủ môn

```
soup_keepers = bs(html_keepers, 'html.parser')
div_keepers = soup_keepers.find('div', id='div_stats_keeper')
table_keepers = div_keepers.find('table', id='stats_keeper')
body_keepers = table_keepers.find('tbody')
rows_keepers = body_keepers.find_all('tr')

keepers_data = []

for row in rows_keepers:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in keeper_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val

    try:
        minutes = int(player_data.get('gk_minutes', '0').replace(',', ''))
        if minutes >= 90:
            keepers_data.append(player_data)
    except:
        continue
```

➔ Dùng BeautifulSoup để phân tích HTML -> tìm bảng stats_keeper
➔ Lấy giữ liệu mỗi dòng, chỉ giữ lại các thủ môn có thời gian chơi trên 90p

```
df_keepers = pd.DataFrame(keepers_data)
```

➔ Tạo Data Frame từ danh sách keepers_data

3. Lấy dữ liệu Shooting:

```
url_shooting = "https://fbref.com/en/comps/9/shooting/Premier-League-Stats"
```

➔ URL của shooting

```python
options3 = uc.ChromeOptions()
options3.add_argument("--no-sandbox")
options3.add_argument("--disable-dev-shm-usage")
options3.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options3, headless=False) as driver:
    driver.get(url_shooting)
    time.sleep(10)
    html_shooting = driver.page_source

shooting_columns = {
    'player': 'Player',
    'shots_on_target_pct': 'SoT%',
    'shots_on_target_per90': 'SoT/90',
    'goals_per_shot': 'G/Sh',
    'average_shot_distance': 'Dist'
}
```

➔ Mở trình duyệt, chờ 10 giây, lấy mã HTML

➔ Khai báo các chỉ số và tên của cột tương ứng trong DataFrame

```
soup_shooting = bs(html_shooting, 'html.parser')
div_shooting = soup_shooting.find('div', id='div_stats_shooting')
table_shooting = div_shooting.find('table', id='stats_shooting')
tbody_shooting = table_shooting.find('tbody')
rows_shooting = tbody_shooting.find_all('tr')

shooting_data = []

for row in rows_shooting:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in shooting_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val

    try:
        shooting_data.append(player_data)
    except:
        continue
```

➔ Dùng BeautifulSoup để phân tích HTML -> tìm bảng stat_shooting
➔ Lấy các cầu thủ có thời gian chơi trên 90p

```
df_shooting = pd.DataFrame(shooting_data)
```

➔ Tạo Data Frame từ danh sách shooting_data
4. Lấy dữ liệu Passing:

```
url_passing = 'https://fbref.com/en/comps/9/passing/Premier-League-Stats'
```

➔ Truy cập vào trang stat passing

```python
options4 = uc.ChromeOptions()
options4.add_argument("--no-sandbox")
options4.add_argument("--disable-dev-shm-usage")
options4.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options4, headless=False) as driver:
    driver.get(url_passing)
    time.sleep(10)
    html_passing = driver.page_source

passing_columns = {
    'player': 'Player',
    'passes_completed': 'total passes completed (Cmp)',
    'passes_pct': 'Total pass completion (Cmp%)',
    'passes_total_distance': ' progressive passing distance (TotDist)',
    'passes_pct_short': 'Short pass completion (Cmp%)',
    'passes_pct_medium': 'Medium pass completion (Cmp%)',
    'passes_pct_long': 'Long pass completion (Cmp%)',
    'assisted_shots': 'KP',
    'passes_into_final_third': 'pass into final third',
    'passes_into_penalty_area': 'PPA',
    'crosses_into_penalty_area': 'CrsPA',
    'progressive_passes': 'PrgP'
}
```

➔ Mở trình duyệt, chờ 10s để tải trang, lấy HTML

➔ Khai báo các chỉ số và tên của cột tương ứng trong DataFrame

```
soup_passing = bs(html_passing, 'html.parser')
div_passing = soup_passing.find('div', id='div_stats_passing')
table_passing = div_passing.find('table', id='stats_passing')
tbody_passing = table_passing.find('tbody')
rows_passing = tbody_passing.find_all('tr')

passing_data = []

for row in rows_passing:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in passing_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val
    try:
        passing_data.append(player_data)
    except:
        continue

df_passing = pd.DataFrame(passing_data)
```

➔ Dùng BeautifulSoup phân tích HTML -> tìm bảng stat_passing
➔ Lấy dữ liệu từ bảng với các chỉ số đã khai báo
➔ Tạo Data Frame từ danh sách passing_data

5. Lấy dữ liệu Goal Shot Creation:

```python
url_GoalShot = 'https://fbref.com/en/comps/9/gca/Premier-League-Stats'

options5 = uc.ChromeOptions()
options5.add_argument("--no-sandbox")
options5.add_argument("--disable-dev-shm-usage")
options5.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options5, headless=False) as driver:
    driver.get(url_GoalShot)
    time.sleep(10)
    html_GoalShot = driver.page_source

GoalShot_columns = {
    'player': 'Player',
    'sca': ' SCA',
    'sca_per90': 'SCA90',
    'gca': 'GCA',
    'gca_per90': 'GCA90',
}
```

➔ Truy cập vào trang Goal and Shot Creation
➔ Mở trình duyệt, chờ 10s để tải trang, lấy mã HTML
➔ Khai báo các chỉ số và tên của cột tương ứng trong DataFrame

```
soup_GoalShot = bs(html_GoalShot, 'html.parser')
div_GoalShot = soup_GoalShot.find('div', id='div_stats_gca')
table_GoalShot = div_GoalShot.find('table', id='stats_gca')
tbody_GoalShot = table_GoalShot.find('tbody')
rows_GoalShot = tbody_GoalShot.find_all('tr')

# Thu thập dữ liệu GoalShot    "thập": Unknown word.
GoalShot_data = []

for row in rows_GoalShot:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in GoalShot_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val

    try:
        GoalShot_data.append(player_data)
    except:
        continue

df_GoalShot = pd.DataFrame(GoalShot_data)
```

➔ Dùng BeautifulSoup để phân tích HTML -> tìm bảng stat_gca
➔ Lấy dữ liệu từ mỗi dòng bảng và xử lý
➔ Tạo DataFrame từ danh sách GoalShot_data

6. Lấy dữ liệu Defense Stat:

```python
url_Defense = 'https://fbref.com/en/comps/9/defense/Premier-League-Stats'

options6 = uc.ChromeOptions()
options6.add_argument("--no-sandbox")
options6.add_argument("--disable-dev-shm-usage")
options6.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options6, headless=False) as driver:
    driver.get(url_Defense)
    time.sleep(10)
    html_Defense = driver.page_source

Defense_columns = {
    'player': 'Player',
    'tackles': 'Def_Tkl',
    'tackles_won': 'Def_TklW',
    'challenges': 'Def_Att',
    'blocks': 'Def_Blocks',
    'blocked_shots': 'Def_Sh',
    'blocked_passes': 'Def_Pass',
    'interceptions': 'Def_Int'
}
```

➔ Truy cập vào trang Defense Stat
➔ Mở trình duyệt, chờ 10s để tải trang, lấy mã HTML
➔ Khai báo các chỉ số và tên cột tương ứng trong DataFrame

```
Defense_data = []

for row in rows_Defense:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in Defense_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val

    try:
        Defense_data.append(player_data)
    except:
        continue

df_Defense = pd.DataFrame(Defense_data)
```

➔ Lấy dữ liệu từ mỗi dòng của bảng và xử lý
➔ Tạo DataFrame từ danh sách Defense_data

7. Lấy dữ liệu Possession Stat

```
url_Possession = 'https://fbref.com/en/comps/9/possession/Premier-League-Stats'

options7 = uc.ChromeOptions()
options7.add_argument("--no-sandbox")
options7.add_argument("--disable-dev-shm-usage")
options7.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options7, headless=False) as driver:
    driver.get(url_Possession)
    time.sleep(10)
    html_Possession = driver.page_source

Possession_columns = {
    'player': 'Player', 'touches': ' Touches', 'touches_def_pen_area': 'Def Pen', 'touches_def_3rd': 'Def 3rd',
    'touches_mid_3rd': 'Mid 3rd', 'touches_att_3rd': 'Att 3rd', 'touches_att_pen_area': 'Att Pen ',
    'take_ons': 'Att', 'take_ons_won_pct': 'Succ%', 'take_ons_tackled_pct': 'Tkld%',    "Succ": Unknown word.
    'carries': 'Carries', 'carries_progressive_distance': 'ProDist',
    'progressive_carries': 'ProgC', 'carries_into_final_third': 'Carries 1/3',
    'carries_into_penalty_area': 'CPA', 'miscontrols': 'Mis',    "miscontrols": Unknown word.
    'dispossessed': 'Dis', 'passes_received': 'Rec',
    'progressive_passes_received': 'PrgR ',
}
```

➔ Truy cập vào trang Possession Stat
➔ Mở trình duyệt, chờ 10s để tải trang, lấy mã HTML
➔ Khai báo các chỉ số và tên cột tương ứng trong DataFrame

```
soup_Possession = bs(html_Possession, 'html.parser')
div_Possession = soup_Possession.find('div', id='div_stats_possession')
table_Possession = div_Possession.find('table', id='stats_possession')
tbody_Possession = table_Possession.find('tbody')
rows_Possession = tbody_Possession.find_all('tr')

Possession_data = []

for row in rows_Possession:
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in Possession_columns:
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val

    try:
        Possession_data.append(player_data)
    except:
        continue

df_Possession = pd.DataFrame(Possession_data)
```

➔ Lấy dữ liệu từ mỗi dòng của bảng và xử lý
➔ Tạo DataFrame từ danh sách Possession_data

8. Lấy dữ liệu Miscellaneous Stat:

```
url_Miscel_Stats = 'https://fbref.com/en/comps/9/misc/Premier-League-Stats'

options7 = uc.ChromeOptions()
options7.add_argument("--no-sandbox")
options7.add_argument("--disable-dev-shm-usage")
options7.add_argument("--disable-blink-features=AutomationControlled")

with uc.Chrome(options=options7, headless=False) as driver:
    driver.get(url_Miscel_Stats)     "Miscel": Unknown word.
    time.sleep(10)
    html_Miscel_Stats = driver.page_source     "Miscel": Unknown word.

Miscel_Stats_columns = {    "Miscel": Unknown word.
    'player': 'Player',
    'fouls': 'Fls',
    'fouled': 'Fld',
    'offsides': 'Off',
    'crosses': 'Crs',
    'ball_recoveries': 'Recov',    "Recov": Unknown word.
    'aerials_won': 'Duel_Won',
    'aerials_lost': 'Duel_Lost',
    'aerials_won_pct': 'Duel_Won%',
}
```

➔ Truy cập vào trang Miscellaneous Stat
➔ Mở trình duyệt, chờ 10s để tải trang, lấy mã HTML
➔ Khai báo các chỉ số và tên cột tương ứng trong DataFrame

```
soup_Miscel_Stats = bs(html_Miscel_Stats, 'html.parser')    "Miscel": Unknown word.
div_Miscel_Stats = soup_Miscel_Stats.find('div', id='div_stats_misc')    "Miscel": U
table_Miscel_Stats = div_Miscel_Stats.find('table', id='stats_misc')    "Miscel": Un
tbody_Miscel_Stats = table_Miscel_Stats.find('tbody')    "Miscel": Unknown word.
rows_Miscel_Stats = tbody_Miscel_Stats.find_all('tr')    "Miscel": Unknown word.

Miscel_Stats_data = []    "Miscel": Unknown word.

for row in rows_Miscel_Stats:    "Miscel": Unknown word.
    player_data = {}
    cols = row.find_all('td')
    if not cols:
        continue

    for col in cols:
        stat = col.get('data-stat')
        if stat in Miscel_Stats_columns:    "Miscel": Unknown word.
            val = col.text.strip()
            if val == '':
                val = 'N/a'
            player_data[stat] = val
    try:
        Miscel_Stats_data.append(player_data)    "Miscel": Unknown word.
    except:
        continue

df_Miscel_Stats = pd.DataFrame(Miscel_Stats_data)    "Miscel": Unknown word.
```

➔ Lấy dữ liệu từ mỗi dòng của bảng và xử lý

➔ Tạo DataFrame từ danh sách Miscel_Stats_data

```
df_keepers.rename(columns=keeper_columns, inplace=True)
df_players.rename(columns=player_columns, inplace=True)
df_shooting.rename(columns=shooting_columns, inplace=True)
df_passing.rename(columns=passing_columns, inplace=True)
df_GoalShot.rename(columns=GoalShot_columns, inplace=True)
df_Defense.rename(columns=Defense_columns, inplace=True)
df_Possession.rename(columns=Possession_columns, inplace=True)
df_Miscel_Stats.rename(columns=Miscel_Stats_columns, inplace=True)
```

➔ Đổi tên các cột theo chuẩn để gộp dữ liệu.

```
df_keepers = df_keepers.drop_duplicates(subset=["Player"])
df_shooting = df_shooting.drop_duplicates(subset=["Player"])
df_passing = df_passing.drop_duplicates(subset=["Player"])
df_GoalShot = df_GoalShot.drop_duplicates(subset=["Player"])
df_Defense = df_Defense.drop_duplicates(subset=["Player"])
df_Possession = df_Possession.drop_duplicates(subset=["Player"])
df_Miscel_Stats = df_Miscel_Stats.drop_duplicates(subset=["Player"])
```

➔ Xoá các trùng lặp theo tên (Player)

```
merged_df = pd.merge(df_players, df_keepers, on="Player", how="left")
merged_df = pd.merge(merged_df, df_shooting, on="Player", how="left")
merged_df = pd.merge(merged_df, df_passing, on="Player", how="left")
merged_df = pd.merge(merged_df, df_GoalShot, on="Player", how="left")
merged_df = pd.merge(merged_df, df_Defense, on="Player", how="left")
merged_df = pd.merge(merged_df, df_Possession, on="Player", how="left")
merged_df = pd.merge(merged_df, df_Miscel_Stats, on="Player", how="left")
```

➔ Lần lượt ghép các bảng dữ liệu lại với nhau

```
merged_df['First_Name'] = merged_df['Player'].apply(lambda x: x.split()[0] if isinstance(x, str) else '')
merged_df.sort_values(by='First_Name', inplace=True)
merged_df.drop(columns=['First_Name'], inplace=True)
```

➔ Sắp xếp lại các cầu thủ theo First Name

```
merged_df.fillna("N/a", inplace=True)
merged_df.to_csv("BTL1.csv", index=False)
```

➔ Điền “N/a” vào các khoảng trống

➔ In ra file csv có tên BTL1

II. Bài tập lớn 2:

```
from bs4 import BeautifulSoup
import pandas as pd
import matplotlib.pyplot as plt
import os
from collections import Counter
```

Khai báo các thư viện cần thiết

1. Phần1: Tính toán và in ra file csv:

```
8
9   df = pd.read_csv("premier_league_players_full.csv")
10  numeric_cols = [col for col in df.columns if pd.api.types.is_numeric_dtype(df[col])]
11
12  df['Team'] = df['Team'].str.strip().str.lower().str.title()
13
14  medians = df[numeric_cols].median().to_frame().T
15  medians.index = ["all"]
16  means_all = df[numeric_cols].mean().to_frame().T
17  means_all.index = ["all"]
18  stds_all = df[numeric_cols].std().to_frame().T    "stds": Unknown word.
19  stds_all.index = ["all"]    "stds": Unknown word.
20
21  means_team = df.groupby("Team")[numeric_cols].mean()
22  stds_team = df.groupby("Team")[numeric_cols].std()    "stds": Unknown word.
23  medians_team = df.groupby("Team")[numeric_cols].median()
24
25  # Hàm tạo tên cột
26  def format_cols(stat_type, cols):
27      return [f"{stat_type} of {col}" for col in cols]
28
```

➔ Dòng 9: Đọc dữ liệu từ file CSV chứa thông tin cầu thủ.
➔ Dòng 10: Lấy danh sách các cột có kiểu dữ liệu số (numeric) để tính toán thống kê.
➔ Dòng 12: Chuẩn hóa tên đội bóng: xóa khoảng trắng thừa, chuyển về chữ thường, rồi viết hoa từng chữ cái đầu.
➔ Dòng 14 -19: Tính trung vị, trung bình, và độ lệch chuẩn cho toàn giải.
➔ Dòng 21-23: Tính các chỉ số trên theo từng đội bóng.
➔ Dòng 21-23: Hàm tạo tên cột

```
29  # Hàm tạo DataFrame cho mỗi nhóm    "nhóm": Unknown word.
30  def prepare_stat_df(stat_df, stat_type):
31      stat_df.columns = format_cols(stat_type, stat_df.columns)
32      stat_df.insert(0, "Team", stat_df.index)
33      return stat_df.reset_index(drop=True)
34
35  # Tạo bảng cho dòng "all"    "bảng": Unknown word.
36  df_all = pd.concat([
37      prepare_stat_df(medians, "Median"),
38      prepare_stat_df(means_all, "Mean"),
39      prepare_stat_df(stds_all, "Std")    "stds": Unknown word.
40  ], axis=1)
41
42  df_all = df_all.loc[:, ~df_all.columns.duplicated()]
43  df_all = df_all.iloc[:1]
44
45  # Tạo bảng cho các đội    "bảng": Unknown word.
46  df_team = pd.concat([
47      prepare_stat_df(means_team, "Mean"),
48      prepare_stat_df(stds_team, "Std")    "stds": Unknown word.
49  ], axis=1)
```

➔ Dòng 30-33: Hàm tạo DataFrame cho mỗi nhóm
➔ Dòng 36-40: Tạo bảng cho dòng “all”
➔ Dòng 46 -49: Tạo bảng cho các đội

```
55  # Sắp xếp thứ tự cột theo CSV gốc    "theo": Unknown word.
56  ordered_cols = []
57  for col in numeric_cols:
58      ordered_cols += [
59          f"Median of {col}",
60          f"Mean of {col}",
61          f"Std of {col}"
62      ]
63
64  final_df = pd.concat([df_all, df_team], ignore_index=True)
65  final_df = final_df[["Team"] + ordered_cols]
66  final_df.to_csv("results2.csv", index=False)
67
```

➔ Dòng 56-61: Sắp xếp thứ tự cột theo csv gốc

➔ Dòng 64-66: Gộp bảng “all” và bảng “team” -> Sắp xếp lại các cột cho đúng thứ tự -> xuất ra file csv tên “results2.csv”

2. Vẽ Histogram:

```
70  os.makedirs("histograms", exist_ok=True)
71
72  # Vẽ cho toàn giải    "toàn": Unknown word.
73  for col in numeric_cols:
74      plt.figure(figsize=(8, 5))    "figsize": Unknown word.
75      plt.hist(df[col].dropna(), bins=20, color='skyblue', edgecolor='black')    "skyblue": Unknown word.
76      plt.title(f'Distribution of {col} (All Players)')
77      plt.xlabel(col)    "xlabel": Unknown word.
78      plt.ylabel('Number of Players')    "ylabel": Unknown word.
79      plt.grid(True)
80      plt.tight_layout()
81      safe_col = col.replace("/", "_").replace("\\", "_").replace(":", "_")
82      plt.savefig(f"histograms/all_{safe_col}.png")
83      plt.close()
84
85  # Vẽ cho từng đội    "từng": Unknown word.
86  teams = df['Team'].dropna().unique()
87  for team in teams:
88      team_df = df[df['Team'] == team]
89      for col in numeric_cols:
90          plt.figure(figsize=(8, 5))    "figsize": Unknown word.
91          plt.hist(team_df[col].dropna(), bins=20, color='lightgreen', edgecolor='black')    "lightgreen": Unknown word.
92          plt.title(f'Distribution of {col} ({team})')
93          plt.xlabel(col)    "xlabel": Unknown word.
94          plt.ylabel('Number of Players')    "ylabel": Unknown word.
95          plt.grid(True)
96          plt.tight_layout()
97          team_filename = team.replace(" ", "_")
98          safe_col = col.replace("/", "_").replace("\\", "_").replace(":", "_")
99          plt.savefig(f"histograms/{team_filename}_{safe_col}.png")
100         plt.close()
```

➔ Dòng 70: Tạo thư mục Histogram nếu chưa tồn tại

➔ Dòng 73-83 **(Vẽ Histogram cho toàn giải)**: Lặp qua từng chỉ số, vẽ histogram bằng matplotlib.pyplot và lưu lại thành ảnh dưới định dạng .png

➔ Dòng 86-100 **(Vẽ Histogram cho từng đội):** Tương tự như toàn giải, sử dụng điều kiện lọc df[df['Team'] == team] để vẽ histogram chỉ cho cầu thủ thuộc đội đó.

3. Tìm ra đội mạnh nhất:

```
105  team_means = df.groupby("Team")[numeric_cols].mean()
106
107  # Xác định đội đứng đầu từng chỉ số    "định": Unknown word.
108  top_teams = {}
109  for col in numeric_cols:
110      top_team = team_means[col].idxmax()
111      top_score = team_means[col].max()
112      top_teams[col] = (top_team, top_score)
113
114  # Đếm số lần mỗi đội đứng đầu    "đứng": Unknown word.
115  top_counts = Counter(team for team, _ in top_teams.values())
116  best_team = top_counts.most_common(1)[0]
117
118  # Ghi kết quả ra file
119  with open("best_team_stats.txt", "w", encoding="utf-8") as f:
120      f.write("Top team for each statistic:\n")
121      for col, (team, score) in top_teams.items():
122          f.write(f"{col}: {team} ({score:.2f})\n")
123      f.write("\nTeam with the most top stats:\n")
124      f.write(f"{best_team[0]} with {best_team[1]} top scores.\n")
```

➔ Dòng 105: Tính trung bình các chỉ số theo đội
➔ Dòng 108-112: Xác định các đội đứng đầu từng chỉ số
➔ Dòng 114-116: Dếm số lần đứng đầu của mỗi đội
➔ Dòng 119-124: Ghi kết quả vào file “best_team_stats.txt”

III. Bài tập lớn 3:

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  from sklearn.preprocessing import StandardScaler
4  from sklearn.decomposition import PCA
5  from sklearn.cluster import KMeans
6  from sklearn.metrics import silhouette_score
```

➔ Khai báo các thư viện cần thiết

```
8   df = pd.read_csv("premier_league_players_full.csv")
9
10  # Lọc các cột số học
11  numeric_cols = [col for col in df.columns if pd.api.types.is_numeric_dtype(df[col])]
12  df_numeric = df[numeric_cols].dropna()  # Loại bỏ các hàng có giá trị thiếu    "Loại": L
13
14  # Chuẩn hóa dữ liệu    "Chuẩn": Unknown word.
15  scaler = StandardScaler()
16  df_scaled = scaler.fit_transform(df_numeric)
17
18  #Tìm số cụm tối ưu với Elbow Method
19  inertias = []
20  k_range = range(1, 11)
21
22  for k in k_range:
23      kmeans = KMeans(n_clusters=k, random_state=42)    "kmeans": Unknown word.
24      kmeans.fit(df_scaled)    "kmeans": Unknown word.
25      inertias.append(kmeans.inertia_)    "kmeans": Unknown word.
26
27  plt.figure(figsize=(8, 4))    "figsize": Unknown word.
28  plt.plot(k_range, inertias, marker='o')
29  plt.title("Elbow Method for Optimal k")
30  plt.xlabel("Number of clusters (k)")    "xlabel": Unknown word.
31  plt.ylabel("Inertia")    "ylabel": Unknown word.
32  plt.grid(True)
33  plt.tight_layout()
34  plt.savefig("elbow_plot.png")
35  plt.show()
```

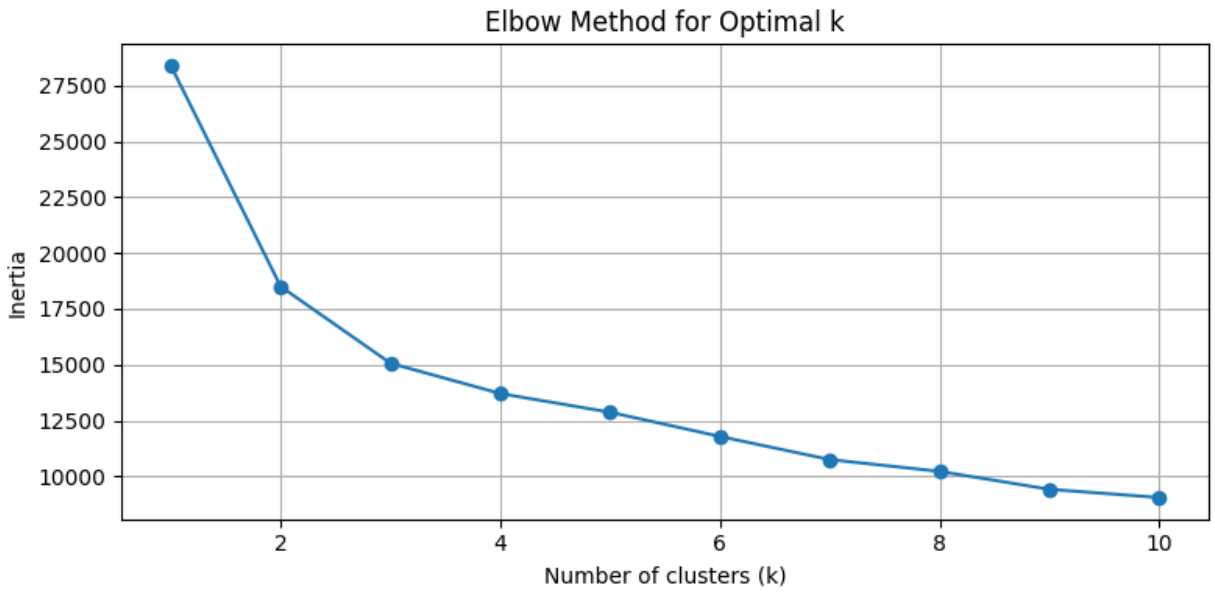
➔ Dòng 8: Đọc dữ liệu thống kê cầu thủ từ file CSV.

➔ Dòng 11-12: Lọc ra các cột có giá trị số (numeric) để dùng cho phân cụm, bỏ các hang có giá trị tối thiểu.

➔ Dòng 15-16: Chuẩn hóa dữ liệu về cùng thang đo (trung bình = 0, độ lệch chuẩn = 1)

➔ Dòng 19-25: Ta chạy thử K-means các giá trị từ 1->10. Tính tổng bình phương khoảng cách từ điểm đến tâm cụm, để xác định độ “chặt” của cụm

➔ Dòng 27-35: Vẽ biểu đồ Elbow Plot để tìm "k điểm gấp khúc", là số cụm tối ưu (optimal_k)



➔ Dựa vào biểu đồ Elbow, ta chọn được k = 3 là điểm gấp khúc

```
#Áp dụng KMeans với số cụm tối ưu    "dụng": Unknown word
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(df_scaled)    "kmeans": Unk
df['Cluster'] = clusters
```

➔ Chạy phân cụm KMeans với số cụm tối ưu và gán nhãn cụm vào dataframe.

```
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(df_scaled)
df['PCA1'] = reduced_data[:, 0]
df['PCA2'] = reduced_data[:, 1]
```

➔ Giảm chiều dữ liệu với PCA để vẽ biểu đồ

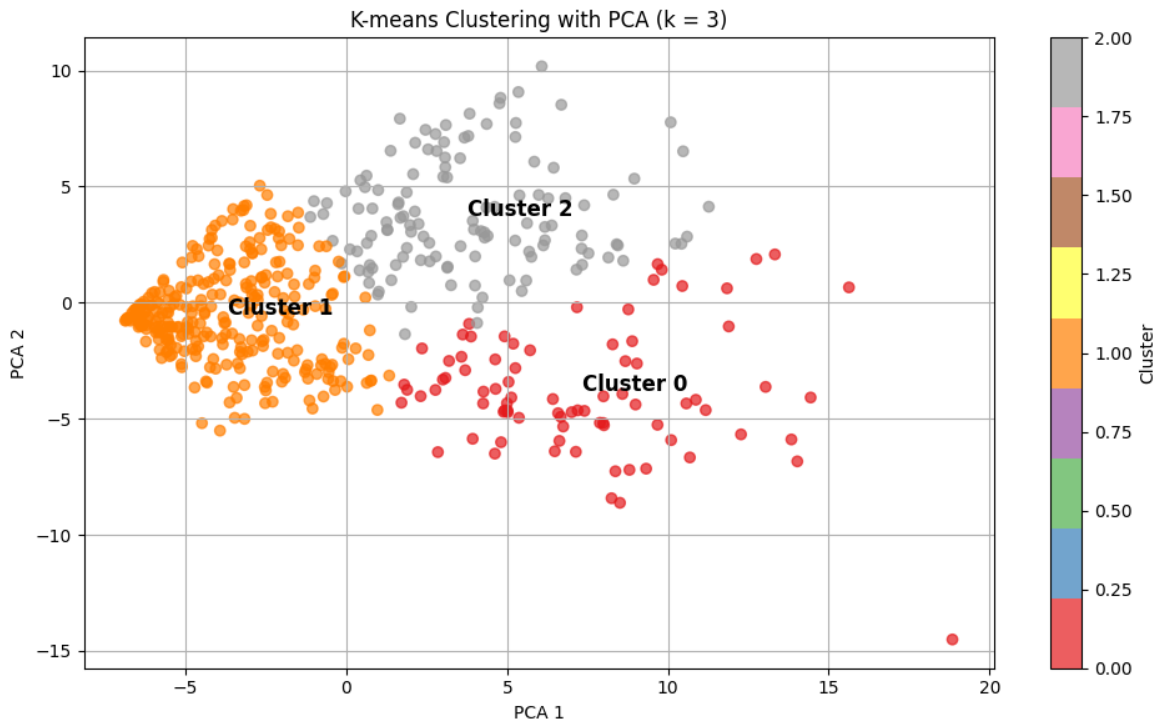
```
50  plt.figure(figsize=(10, 6))    "figsize": Unknown word.
51  scatter = plt.scatter(df['PCA1'], df['PCA2'], c=df['Cluster'], cmap='Set1', alpha=0.7)    "cmap": Unknown wor
52
53  # Thêm nhãn cụm    "Thêm": Unknown word.
54  for i in range(optimal_k):
55      cluster_center = df[df['Cluster'] == i][['PCA1', 'PCA2']].mean()
56      plt.text(cluster_center['PCA1'], cluster_center['PCA2'], f'Cluster {i}', fontsize=12, weight='bold')
57
58  plt.title(f"K-means Clustering with PCA (k = {optimal_k})")
59  plt.xlabel("PCA 1")    "xlabel": Unknown word.
60  plt.ylabel("PCA 2")    "ylabel": Unknown word.
61  plt.colorbar(scatter, label='Cluster')    "colorbar": Unknown word.
62  plt.grid(True)
63  plt.tight_layout()
64  plt.savefig("cluster_pca_plot.png")
65  plt.show()
66
67  #Lưu kết quả
68  df.to_csv("clusters_with_pca.csv", index=False)
```

➔ Dòng 50-51: Tạo một hình mới với kích thước 10 inch × 6 inch.
Vẽ các điểm theo trục hoành là PCA1, trục tung là PCA2, tô màu theo nhãn cụm(Cluster), dùng bảng màu "Set1", alpha = 0.7 -> làm cho điểm mơ đi để dễ nhìn thấy các điểm trông lên nhau

➔ Dòng 54-56: Thêm nhãn cụm vào vị trí trung tâm của từng cụm.

➔ Dòng 58-65:

- Đặt tiêu đề cho biểu đồ là: **"K-means Clustering with PCA (k =3)"**
- Gắn trục x là PCA1
- Gắn trục y là PCA2
- Thêm màu vào bên cạnh để biểu diễn ý nghĩa màu: mỗi màu ứng với 1 cụm (cluster)
- Bật lưới nền giúp dễ nhìn hơn.
- Tự động **canh chỉnh bố cục** để không bị chồng chữ, sát mép.
- Lưu biểu đồ thành file ảnh PNG với tên "**cluster_pca_plot.png**"
- Hiển thị biểu đồ ra màn hình.



- Cluster 0: Các hậu vệ/ tiền vệ phòng ngự/tiền vệ trung tâm(ít bàn, ít kiến tạo, tắc bóng nhiều)
- Cluster 1: Các tiền vệ trung tâm / tiền vệ công/ tiền đạo (ghi nhiều bàn, kiến tạo cao, nhiều đường chuyền)
- Cluster 2: Các thủ môn (chỉ số cứu thua cao, ít tham gia tấn công)

➔ Dòng 68: Lưu toàn bộ bảng dữ liệu (gồm cột cụm và PCA) vào file CSV để phân tích tiếp.

IV. Bài tập lớn 4:

1. Phần 1:

```
1   import undetected_chromedriver as uc    "chromedriver": Unknown wor
2   from bs4 import BeautifulSoup as bs
3   import pandas as pd
4   from selenium.webdriver.common.by import By
5   from selenium.webdriver.support.ui import WebDriverWait
6   from selenium.webdriver.support import expected_conditions as EC
7   import time
```

➔ Khai báo các thư viện cần thiết

```
9   existing_players_df = pd.read_csv("premier_league_players_full.csv")
10  existing_players = set(existing_players_df['Player'].str.lower().str.strip())
```

➔ Đọc file CSV đã có chứa danh sách đầy đủ cầu thủ
➔ Tạo tập hợp tên cầu thủ (existing_players) đã được chuyển về chữ thường, xóa khoảng trắng — để so sánh tên dễ dàng hơn khi lấy từ web.

```
14    options = uc.ChromeOptions()
15    options.add_argument("--no-sandbox")
16    options.add_argument("--disable-dev-shm-usage")
17    options.add_argument("--disable-blink-features=AutomationControlled")
```

➔ Tạo tùy chọn trình duyệt (Chrome) sao cho trình duyệt tránh bị phát hiện là bot tự động

```
19  with uc.Chrome(options=options, headless=False) as driver:
20      for page in range(1, 23):
21          url = f"https://www.footballtransfers.com/en/values/players/most-valuable-players/playing-in-uk-premier-league/{page}"
22          driver.get(url)
```

➔ Lặp qua 22 trang của bảng giá trị cầu thủ Premier League.
➔ Truy cập vào từng trang bằng trình duyệt tự động (driver.get(url)).

```
try:
    WebDriverWait(driver, 15).until(
        EC.presence_of_element_located((By.TAG_NAME, "table"))
    )
except Exception:
    continue
```

➔ Sử dụng WebDriverWait để đợi bảng dữ liệu HTML xuất hiện trong tối đa 15 giây trước khi phân tích.

```
31          html = driver.page_source
32          soup = bs(html, 'html.parser')
33          table = soup.find('table', class_='table table-hover no-cursor table-striped leaguetable mvp-table mb-0')
34
35          if not table:
36              continue
37
38          rows = table.find('tbody').find_all('tr')
39          for row in rows:
40              player_td = row.find('td', class_='td-player')
41              row_value = row.find_all('td', class_='text-center')
42              value_td = row_value[1].find('span', class_='player-tag') if len(row_value) > 1 else None
43
44              if player_td:
45                  name_tag = player_td.find('a')
46                  player_name = name_tag.get_text(strip=True).lower() if name_tag else 'n/a'
47                  value = value_td.get_text(strip=True) if value_td else 'n/a'
48
49                  if player_name in existing_players:
50                      data.append({
51                          'Player': player_name.title(),
52                          'Transfer Value': value
53                      })
54
55          time.sleep(5)  # Giúp tránh bị chặn IP    "Giúp": Unknown word.
```

➔ Dòng 31-32: Lấy mã HTML của trang hiện tại, phân tích bằng BeautifulSoup

➔ Dòng 33: Tìm bảng HTML chứa dữ liệu cần lấy dựa theo class định danh.

➔ Dòng 38: Lấy tất cả hàng (mỗi cầu thủ một hàng).

➔ Dòng 39-42: Duyệt qua các thẻ ”tr”, lấy phần chứa tên các cầu thủ và giá trị chuyển nhượng

➔ Dòng 44-53: Chuyển “name” về định dạng thường để so sánh

Nếu tên cầu thủ tồn tại trong tập existing_players, thì thêm vào danh sách data(chuyển lại title() cho đẹp)

Transfer Value là giá trị

➔ Dòng 55: Tạm dừng 5s để tránh nguy cơ bị chặn

```
58  df = pd.DataFrame(data)
59  df.to_csv("BTL4.csv", index=False)
```

➔ Tạo DataFrame để lưu danh sách “data” và in ra file csv tên “BTL4.csv”

2. Phần 2:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import logging
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.stats import spearmanr, randint, uniform    "spearmanr": Unknown word.
from math import sqrt
import xgboost as xgb
```

➔ Khai báo các thư viện cần thiết

```python
warnings.filterwarnings("ignore")
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
logging.getLogger('sklearn').setLevel(logging.CRITICAL)
plt.style.use('ggplot')    "ggplot": Unknown word.
```

➔ Giảm thiểu sự nhiễu loạn đầu ra khi chạy mô hình, giúp hiển thị kết quả rõ rang

```python
def load_data():
    features = pd.read_csv("BTL1.csv")
    values = pd.read_csv("BTL4.csv")
    for df in [features, values]:
        df["Player"] = df["Player"].str.strip().str.lower()
    merged = pd.merge(features, values, on="Player", how="inner", suffixes=('', '_drop'))
    merged = merged.filter(regex='^(?!.*_drop)')
    return merged
```

➔ Nhiệm vụ tải, chuẩn hóa và hợp nhất dữ liệu từ hai file CSV liên quan đến cầu thủ: đặc trưng hiệu suất và giá trị chuyển nhượng

```python
def preprocess_data(df):
    def convert_value(value):
        try:
            value = str(value).lower().replace("\u20ac", "").replace(",", "").strip()
            if 'm' in value: return float(value.replace('m', '')) * 1e6
            if 'k' in value: return float(value.replace('k', '')) * 1e3
            return float(value)
        except:
            return np.nan

    df["Transfer Value"] = df["Transfer Value"].apply(convert_value)
    df = df[df["Transfer Value"] > 1000]

    num_cols = df.select_dtypes(include=np.number).columns.tolist()
    extra_cols = ['Goal', 'Assists', 'Age']
    for col in extra_cols:
        if col in df.columns and col not in num_cols:
            df[col] = pd.to_numeric(df[col], errors='coerce')
            num_cols.append(col)

    df['Goal_per_Age'] = df['Goal'] / df['Age']
    df['Assists_per_Age'] = df['Assists'] / df['Age']
    df['Log_Transfer_Value'] = np.log1p(df['Transfer Value'])

    df[['Goal_per_Age', 'Assists_per_Age']] = df[['Goal_per_Age', 'Assists_per_Age']].fillna(0)
    df[num_cols] = df[num_cols].fillna(df[num_cols].median())

    return df
```

➔ Chuyển đổi giá trị chuyển nhượng về số (convert_value)
➔ Loại bỏ cầu thủ có giá trị chuyển nhượng < 1,000 euro (lọc nhiễu).
➔ Chuyển đổi các cột Goal, Assists, Age về dạng số nếu cần
➔ Điền khuyết (fillna) bằng trung vị cho các cột số.

```python
def evaluate_model(model, X, y, log_target=True):
    pred = model.predict(X)
    if log_target:
        y_true = np.expm1(y)
        pred = np.expm1(pred)
    else:
        y_true = y

    rmse = sqrt(mean_squared_error(y_true, pred))
    mae = mean_absolute_error(y_true, pred)
    r2 = r2_score(y_true, pred)
    spear_corr, _ = spearmanr(y_true, pred)    "spearmanr": Unknown word.

    print(f"  R2 Score: {r2:.4f}")
    print(f"  RMSE: {rmse/1e6:.4f}")
    print(f"  MAE: {mae/1e6:.4f}")
    print(f"  Spearman: {spear_corr:.4f}")
```

➔ Dùng để đánh giá chất lượng dự đoán của mô hình hồi quy sau khi đã huấn luyện:

➔ Dòng 77: Dự đoán giá trị đầu ra (Transfer Value) từ dữ liệu đầu vào X

➔ Dòng 78-82: Nếu log_target=True (mặc định), có nghĩa là y là log(Transfer Value + 1)

Hàm np.expm1(x) = exp(x) - 1 được dùng để khôi phục lại giá trị gốc từ giá trị log-transformed.

➔ Dòng 84: RMSE: sai số trung bình bình phương căn – càng thấp càng tốt.

➔ Dòng 85: MAE: trung bình khoảng cách tuyệt đối giữa dự đoán và thực tế.

➔ Dòng 86: R^2: đo mức độ mô hình giải thích phương sai trong dữ liệu thực (1 là tốt nhất, 0 là ngẫu nhiên).

➔ Dòng 87: Spearman correlation: đo độ tương quan thứ hạng giữa dự đoán và thực tế.

➔ Dòng 89-92: In kết quả đánh giá

```
95  if __name__ == "__main__":
96      df = load_data()
97      df = preprocess_data(df)
98
99      X = df.drop(columns=['Transfer Value', 'Log_Transfer_Value', 'Player'])
100     y = df['Log_Transfer_Value']
101     numeric_features = X.select_dtypes(include=np.number).columns.tolist()
102
103     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
104
105     model = build_model(numeric_features)
106
107     param_dist = {
108         'regressor__n_estimators': randint(100, 400),
109         'regressor__learning_rate': uniform(0.05, 0.2),
110         'regressor__max_depth': randint(4, 8),
111         'regressor__subsample': uniform(0.7, 0.3),
112         'regressor__colsample_bytree': uniform(0.7, 0.3)    "colsample": Unknown word.
113     }
114
115     search = RandomizedSearchCV(model, param_dist, n_iter=15, cv=3,
116                                 scoring='neg_root_mean_squared_error', random_state=42, n_jobs=-1)
117     search.fit(X_train, y_train)
118
119     best_model = search.best_estimator_
120
121     print("\nTrain:")
122     evaluate_model(best_model, X_train, y_train)
123     print("Test:")
124     evaluate_model(best_model, X_test, y_test)
```

➔ Dòng 95: Khai báo hàm main

➔ Dòng 96-97: Gọi hàm load_data() để tải và hợp nhất dữ liệu từ các file CSV.

Gọi hàm preprocess_data() để chuyển đổi, làm sạch và tạo đặc trưng mới

➔ Dòng 99-100: X: ma trận đặc trưng
Y: biến mục tiêu là giá trị chuyển nhượng đã log hóa (Log_Transfer_Value) để làm trơn phân phối dữ liệu.

➔ Dòng 101: Lọc ra các đặc trưng dạng số để chuẩn hóa (scale) sau này bằng StandardScaler

➔ Dòng 103: Chia dữ liệu thành 80% để huấn luyện và 20% để kiểm tra, random_state=42 giúp tái tạo kết quả.

➔ Dòng 105: Gọi hàm build_model() để tạo pipeline

➔ Dòng 107-113: Xác định không gian tìm kiếm siêu tham số cho mô hình XGBoost

➔ Dòng 115-117: Chạy tối ưu ngẫu nhiên 15 lần để chọn mô hình tốt nhất

➔ Dòng 119: Trích xuất pipeline tốt nhất sau quá trình RandomizedSearchCV.

➔ Dòng 121-124: Gọi hàm evaluate_model() để in ra