BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI BÁO CÁO

MÔN: TOÁN RỜI RẠC 2

Giảng viên: Kim Ngọc Bách

Tên sinh viên: Bùi Quang Thắng

Mã sinh viên: B23DCCN751

Lớp: D23CQCE06 - B

Môn: Lập trình Python

Hà Nội, Tháng 5/2025

Lời mở đầu

Trong bối cảnh công nghệ thông tin và khoa học dữ liệu ngày càng phát triển, Python đã khẳng định vị thế là một trong những ngôn ngữ lập trình mạnh mẽ và phổ biến, đặc biệt trong việc xử lý dữ liệu, phân tích thống kê và học máy. Bài tập lớn này được thực hiện nhằm đáp ứng các yêu cầu của Assignment 1, môn Lập trình Python, với mục tiêu áp dụng kiến thức lý thuyết vào việc thu thập, phân tích và trực quan hóa dữ liệu thống kê của các cầu thủ tham gia mùa giải English Premier League 2024-2025.

Dự án tập trung vào việc thu thập dữ liệu từ các nguồn uy tín, phân tích các chỉ số hiệu suất, phân loại cầu thủ bằng thuật toán học máy, và đề xuất phương pháp định giá cầu thủ dựa trên giá trị chuyển nhượng. Thông qua quá trình thực hiện, chúng tôi không chỉ củng cố kỹ năng lập trình mà còn rèn luyện khả năng tư duy phân tích, giải quyết vấn đề thực tiễn và trình bày kết quả một cách khoa học.

Chúng tôi xin gửi lời cảm ơn chân thành đến giảng viên đã hướng dẫn tận tình, cùng các nguồn tài liệu tham khảo từ `https://foref.com` và `https://www.footballtransfers.com` đã hỗ trợ chúng tôi hoàn thành bài tập. Mọi ý kiến đóng góp sẽ là động lực quý báu để chúng tôi cải thiện và phát triển trong tương lai.

Trân trọng,

Bài 1:

Mã Nguồn Đầy Đủ

Dưới đây là mã nguồn đầy đủ, được định dạng bằng gói `listings` để dễ đọc:

```
# ==========================
# 1. SETUP ON COLAB / LOCAL MACHINE
# ==========================
!pip install selenium
!apt-get update -y
!apt install -y chromium-chromedriver
!cp /usr/lib/chromium-browser/chromedriver /usr/bin

# ==========================
# 2. IMPORT LIBRARIES
# ==========================
import time
import pandas as pd
import re
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

# ==========================
# 3. CONFIGURE SELENIUM CHROME
# ==========================
options = Options()
options.add_argument('--headless')
options.add_argument('--no-sandbox')
options.add_argument('--disable-dev-shm-usage')
driver = webdriver.Chrome(options=options)

# ==========================
# 4. DEFINE TABLES TO SCRAPE
# ==========================
tables = [
    {
        "url":
           "https://fbref.com/en/comps/9/stats/Premier-League-Stats",
        "id": "stats_standard",
        "cols": ["Nation", "Squad", "Position", "Current age",
           "Matches Played", "Minutes",
                 "Goals", "Assists", "Yellow Cards", "Red Cards",
                 "xG: Expected Goals", "xAG: Exp. Assisted Goals",
                 "Progressive Carries", "Progressive Passes",
                    "Progressive Passes Rec",
                 "Goals/90", "Assists/90", "xG/90", "xAG/90"]
    },
    {
        "url":
```

```
            "https://fbref.com/en/comps/9/keepers/Premier-League-Stats",
        "id": "stats_keeper",
        "cols": ["Goals Against/90", "Save Percentage", "Clean
            Sheet Percentage", "Save% (Penalty Kicks)"]
    },
    {
        "url":
            "https://fbref.com/en/comps/9/shooting/Premier-League-Stats",
        "id": "stats_shooting",
        "cols": ["Shots on Target %", "Shots on target/90",
            "Goals/Shot", "Average Shot Distance"]
    },
    {
        "url":
            "https://fbref.com/en/comps/9/passing/Premier-League-Stats",
        "id": "stats_passing",
        "cols": ["Passes Completed", "Pass Completion %", "Total
            Passing Distance",
                "Passes Completed (Short)", "Pass Completion %
                    (Medium)", "Pass Completion % (Long)",
                "Key Passes", "Passes into Final Third", "Passes
                    into Penalty Area",
                "Crosses into Penalty Area", "Progressive
                    Passes"]
    },
    {
        "url":
            "https://fbref.com/en/comps/9/gca/Premier-League-Stats",
        "id": "stats_gca",
        "cols": ["Shot-Creating Actions", "Shot-Creating
            Actions/90",
                "Goal-Creating Actions", "Goal-Creating
                    Actions/90"]
    },
    {
        "url":
            "https://fbref.com/en/comps/9/defense/Premier-League-Stats",
        "id": "stats_defense",
        "cols": ["Tackles", "Tackles Won", "Dribbles Challenged",
            "Challenges Lost",
                "Blocks", "Shots Blocked", "Passes Blocked",
                    "Interceptions"]
    },
    {
        "url":
            "https://fbref.com/en/comps/9/possession/Premier-League-Stats",
        "id": "stats_possession",
        "cols": ["Touches", "Touches (Def Pen)", "Touches (Def
            3rd)", "Touches (Mid 3rd)",
                "Touches (Att 3rd)", "Touches (Att Pen)",
                    "Take-Ons Attempted",
```

```
                    "Successful Take-On %", "Tackled During Take-On Percentage",
                    "Carries", "Progressive Carrying Distance", "Progressive Carries",
                    "Carries into Final Third", "Carries into Penalty Area",
                    "Miscontrols", "Dispossessed", "Passes Received", "Progressive Passes Rec"]
    },
    {
        "url": "https://fbref.com/en/comps/9/misc/Premier-League-Stats",
        "id": "stats_misc",
        "cols": ["Yellow Cards", "Red Cards", "Fouls Committed", "Fouls Drawn",
                    "Offsides", "Crosses", "Ball Recoveries", "Aerials Won",
                    "Aerials Lost", "% of Aerials Won"]
    }
]

# ==========================
# 5. FUNCTION TO SCRAPE A TABLE
# ==========================
def scrape_table(url, table_id, required_cols):
    print(f"   í    Loading table {table_id}     ")
    driver.get(url)
    time.sleep(3)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    tbl = soup.find('table', {'id': table_id})
    if tbl is None:
        print(f"    Table {table_id} not found!")
        return None

    # Header
    header_cells = tbl.find('thead').find_all('tr')[-1].find_all('th')
    headers = []
    for th in header_cells:
        labels = th.get('aria-label') or th.text.strip()
        headers.append(labels)
    headers[0] = 'Player'

    # Data
    rows = []
    for tr in tbl.find('tbody').find_all('tr'):
        if tr.get('class') == ['thead']:
            continue
        cols = tr.find_all(['th','td'])
        if len(cols) != len(headers):
            continue
```

```
        vals = []
        for i, cell in enumerate(cols):
            txt = cell.get_text(strip=True)
            if i == 0:
                txt = re.sub(r'^[\d\.\s]+', '', re.sub(r'[(),]',
                    '', txt)).strip()
            vals.append(txt)
        rows.append(vals)

    df = pd.DataFrame(rows, columns=headers).set_index('Player')
    df = df.add_prefix(f"{table_id}_")

    # Select required columns
    want = [f"{table_id}_{c}" for c in required_cols]
    have = [c for c in want if c in df.columns]
    missing = set(want) - set(have)
    if missing:
        print(f"       Table {table_id} missing columns:
            {sorted(missing)}")
    return df[have]

# ==========================
# 6. SCRAPE ALL TABLES
# ==========================
dfs = {}
for t in tables:
    df = scrape_table(t["url"], t["id"], t["cols"])
    dfs[t["id"]] = df

driver.quit()

# ==========================
# 7. MERGE DATA
# ==========================
std = dfs['stats_standard']
keepers = dfs['stats_keeper']
others = [df for key, df in dfs.items() if key not in
    ['stats_standard','stats_keeper']]

merged = std.copy()
for o in others:
    merged = merged.join(o, how='left')

# GK: Merge keeper stats for goalkeepers
if keepers is not None:
    is_gk = merged['stats_standard_Position'].str.contains('GK',
        na=False)
    for col in keepers.columns:
        merged[col] = pd.NA
        merged.loc[is_gk, col] =
            keepers.reindex(merged.index).loc[is_gk, col]
```

```
165
166 # ==========================
167 # 8. CLEAN PLAYER COLUMN
168 # ==========================
169 merged = merged.reset_index()
170 merged['Player'] = merged['Player'].apply(lambda x: x[1] if
    isinstance(x, tuple) else x)
171 player_col = merged.pop('Player')
172 merged.insert(0, 'Player', player_col)
173
174 # ==========================
175 # 9. FILTER >90 MINUTES
176 # ==========================
177 merged['stats_standard_Minutes'] = (
178     merged['stats_standard_Minutes']
179     .str.replace(',', '', regex=False)
180     .astype(float, errors='ignore')
181 )
182 merged = merged[ merged['stats_standard_Minutes'] > 90 ]
183
184 # ==========================
185 # 10. REMOVE DUPLICATE PLAYERS
186 # ==========================
187 merged = merged.drop_duplicates(subset=['Player'])
188
189 # ==========================
190 # 11. REPLACE EMPTY VALUES WITH 'N/a' (BUT KEEP 0)
191 # ==========================
192 merged = merged.replace(['', '    '], pd.NA)
193 merged = merged.fillna('N/a')
194
195 # ==========================
196 # 12. SORT AND SAVE TO CSV
197 # ==========================
198 merged = merged.sort_values('Player')
199 print(f"Total players: {len(merged)}")
200 merged.to_csv('results.csv', index=False)
201 print(" Results saved to results.csv")
```

Phân Tích Chi Tiết Mã Nguồn

Mã nguồn được chia thành 12 phần chính, mỗi phần có chức năng riêng. Dưới đây là phân tích chi tiết:

1. Cài đặt trên Colab / Máy Local

Phần này cài đặt các thư viện và công cụ cần thiết:

- `!pip install selenium`: Cài đặt thư viện `selenium` để tự động hóa trình duyệt.

- `!apt-get update -y` và `!apt install -y chromium-chromedriver`: Cập nhật danh sách gói và cài đặt ChromeDriver.
- `!cp /usr/lib/chromium-browser/chromedriver /usr/bin`: Sao chép ChromeDriver vào thư mục hệ thống.

2. Import Library

Nhập các thư viện cần thiết:

- `time`: Để thêm độ trễ trong quá trình tải trang web.
- `pandas`: Để xử lý dữ liệu dạng bảng.
- `re`: Để sử dụng biểu thức chính quy làm sạch dữ liệu.
- `BeautifulSoup`: Để phân tích HTML.
- `selenium`: Để điều khiển trình duyệt Chrome.

3. Cấu hình Selenium Chrome

Cấu hình trình duyệt Chrome chạy ở chế độ không giao diện (headless):

- Sử dụng `Options()` để tùy chỉnh Chrome.
- Các tham số như `-headless`, `-no-sandbox`, và `-disable-dev-shm-usage` đảm bảo hoạt động ổn định trên Colab.

4. Định nghĩa các bảng cần lấy

Xác định danh sách các bảng để thu thập dữ liệu:

- Mỗi bảng được định nghĩa bằng một từ điển với `url`, `id`, và `cols`.
- Ví dụ: Bảng `stats_standard` lấy các cột như "Goals", "Assists", "Minutes", v.v.

5. Hàm scrape 1 bảng

Hàm `scrape_table` thu thập dữ liệu từ một bảng:

- Sử dụng Selenium để tải trang và BeautifulSoup để phân tích HTML.
- Trích xuất tiêu đề và dữ liệu, làm sạch tên cầu thủ bằng regex.
- Tạo DataFrame và lọc các cột cần thiết.

6. Scrape toàn bộ

Lặp qua danh sách `tables` để thu thập tất cả các bảng và lưu vào từ điển `dfs`.

7. Gộp dữ liệu

Gộp các DataFrame:

- Bắt đầu với `stats_standard`, sau đó gộp các bảng khác bằng `join`.
- Gộp dữ liệu thủ môn (`stats_keeper`) chỉ cho các cầu thủ có vị trí `GK`.

8. Chỉnh lại cột Player

Làm sạch cột `Player` và di chuyển nó lên đầu DataFrame.

9. Filter >90 phút

Lọc các cầu thủ có thời gian thi đấu trên 90 phút.

10. Xóa trùng Player

Loại bỏ các cầu thủ trùng lặp dựa trên cột `Player`.

11. Thay các giá trị rỗng bằng 'N/a'

Xử lý các giá trị rỗng hoặc không hợp lệ, thay bằng `'N/a'`.

12. Sắp xếp và lưu CSV

Sắp xếp theo tên cầu thủ và lưu kết quả vào `results.csv`.

Bài 2:

Mã Nguồn Đầy Đủ

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import re
import random

# Read data from CSV file
data = pd.read_csv('results.csv')

# Replace "N/a" with NaN
data.replace('N/a', np.nan, inplace=True)

# Identify columns intended to be numeric, excluding irrelevant
    columns
intended_numeric_cols = [
    col for col in data.columns
```

```
    if col != 'stats_standard_Current age'
       and col not in ['Player', 'stats_standard_Nation',
          'stats_standard_Squad', 'stats_standard_Position']
]

# Convert intended numeric columns to numeric type, coercing
   errors to NaN
for col in intended_numeric_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')

# Filter to keep only fully numeric columns
numeric_cols = [col for col in intended_numeric_cols if
   data[col].dtype in ['float64', 'int64']]

# Fill NaN with 0 for numeric columns
data[numeric_cols] = data[numeric_cols].fillna(0)

# Categorize attack and defense metrics
attack_keywords = ['Goals', 'Assists', 'Shots', 'xG', 'xAG']
defense_keywords = ['Tackles', 'Interceptions', 'Blocks',
   'Clearances']
attack_cols = [col for col in numeric_cols if any(keyword in col
   for keyword in attack_keywords)]
defense_cols = [col for col in numeric_cols if any(keyword in col
   for keyword in defense_keywords)]

# Randomly select 3 metrics from each category (or all if fewer
   than 3)
selected_attack_cols = random.sample(attack_cols, min(3,
   len(attack_cols)))
selected_defense_cols = random.sample(defense_cols, min(3,
   len(defense_cols)))

# 1. Identify top 3 highest and lowest for each numeric metric
with open('top_3.txt', 'w', encoding='utf-8') as f:
    for col in numeric_cols:
        f.write(f"Metric: {col}\n")
        f.write("Top 3 highest:\n")
        top_3 = data[['Player',
           col]].dropna().sort_values(by=col,
           ascending=False).head(3)
        for _, row in top_3.iterrows():
            f.write(f"- {row['Player']}: {row[col]:.2f}\n")
        f.write("Top 3 lowest:\n")
        bottom_3 = data[['Player',
           col]].dropna().sort_values(by=col).head(3)
        for _, row in bottom_3.iterrows():
            f.write(f"- {row['Player']}: {row[col]:.2f}\n")
        f.write("\n")

```

```python
# 2. Calculate median, mean, and standard deviation for all players
all_stats = pd.DataFrame({
    '': ['all'],
    'Team': ['all']
})
for col in numeric_cols:
    all_stats[f'Median of {col}'] = [data[col].median()]
    all_stats[f'Mean of {col}'] = [data[col].mean()]
    all_stats[f'Std of {col}'] = [data[col].std()]

# Calculate statistics per team
team_stats = data.groupby('stats_standard_Squad')[numeric_cols].agg(['median', 'mean', 'std']).reset_index()

# Flatten MultiIndex for columns
flattened_columns = [('Team', '')]
for col in numeric_cols:
    flattened_columns.extend([(col, 'median'), (col, 'mean'), (col, 'std')])
team_stats.columns = pd.MultiIndex.from_tuples(flattened_columns)

# Rename columns
team_stats.columns = [f'Median of {col[0]}' if col[1] == 'median' else
                      f'Mean of {col[0]}' if col[1] == 'mean' else
                      f'Std of {col[0]}' if col[1] == 'std' else
                      'Team' for col in team_stats.columns]

# Add index column
team_stats.insert(0, '', range(1, len(team_stats) + 1))

# Ensure matching columns and combine
common_columns = all_stats.columns.intersection(team_stats.columns)
team_stats = team_stats.reindex(columns=common_columns, fill_value=np.nan)
result_stats = pd.concat([all_stats, team_stats], ignore_index=True)
result_stats.to_csv('results2.csv', index=False)

# 3. Identify team with the highest score for each metric
team_totals = data.groupby('stats_standard_Squad')[numeric_cols].sum()
top_teams = team_totals.idxmax()

print("Teams with the highest score for each metric:")
for col, team in top_teams.items():
    print(f"- {col}: {team}")

```

```python
# Evaluate the best performing team based on key metrics
key_stats = [
    'stats_standard_Goals', 'stats_standard_Assists',
    'stats_standard_xG: Expected Goals', 'stats_standard_xAG: Exp. Assisted Goals',
    'stats_keeper_Save Percentage'
]
key_stats = [stat for stat in key_stats if stat in numeric_cols]
team_means = data.groupby('stats_standard_Squad')[key_stats].mean()
team_scores = team_means.sum(axis=1)
best_team = team_scores.idxmax()
best_score = team_scores.max()

print(f"\nBest performing team: {best_team}")
print(
    f"Reason: {best_team} leads with a total average score of {best_score:.2f} across key metrics (goals, assists, xG, xAG, Save%). "
    f"They demonstrate consistent offensive and defensive performance.")

# 4. Plot histograms
# Create histograms directory if it doesn't exist
if not os.path.exists('histograms'):
    os.makedirs('histograms')

# Function to sanitize filenames
def sanitize_filename(name):
    return re.sub(r'[^\w\-_\. ]', '_', name)

# Plot histogram for all players
plt.figure(figsize=(15, 10))
plt.suptitle("Distribution of Selected Metrics for All Players", fontsize=16)
for i, col in enumerate(selected_attack_cols, 1):
    plt.subplot(2, 3, i)
    plt.hist(data[col].dropna(), bins=20, color='blue', alpha=0.7)
    plt.title(f'Attack: {col}')
    plt.xlabel(col)
    plt.ylabel('Count')

for i, col in enumerate(selected_defense_cols, 1):
    plt.subplot(2, 3, i + 3)
    plt.hist(data[col].dropna(), bins=20, color='green', alpha=0.7)
    plt.title(f'Defense: {col}')
    plt.xlabel(col)
    plt.ylabel('Count')

plt.tight_layout(rect=[0, 0, 1, 0.95])
```

```
plt.savefig('histograms/hist_all_players.png')
plt.show()

# Plot histograms for each team
teams = data['stats_standard_Squad'].unique()
for team in teams:
    team_data = data[data['stats_standard_Squad'] == team]
    safe_team = sanitize_filename(team)

    plt.figure(figsize=(15, 10))
    plt.suptitle(f"Distribution of Selected Metrics for {team}",
       fontsize=16)
    for i, col in enumerate(selected_attack_cols, 1):
        plt.subplot(2, 3, i)
        plt.hist(team_data[col].dropna(), bins=20, color='blue',
           alpha=0.7)
        plt.title(f'Attack: {col}')
        plt.xlabel(col)
        plt.ylabel('Count')

    for i, col in enumerate(selected_defense_cols, 1):
        plt.subplot(2, 3, i + 3)
        plt.hist(team_data[col].dropna(), bins=20, color='green',
           alpha=0.7)
        plt.title(f'Defense: {col}')
        plt.xlabel(col)
        plt.ylabel('Count')

    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.savefig(f'histograms/hist_{safe_team}.png')
    plt.show()
```

Phân Tích Chi Tiết Mã Nguồn

Mã nguồn này được chia thành các phần chính, mỗi phần thực hiện một nhiệm vụ cụ thể. Dưới đây là phân tích chi tiết:

1. Nhập thư viện và đọc dữ liệu

Nhập các thư viện cần thiết và đọc dữ liệu từ tệp CSV:

- `pandas`, `numpy`: Để xử lý dữ liệu.
- `matplotlib.pyplot`: Để vẽ biểu đồ.
- `os`, `re`, `random`: Để quản lý tệp, xử lý chuỗi, và chọn ngẫu nhiên.
- Đọc dữ liệu từ `results.csv` vào DataFrame `data`.

2. Làm sạch dữ liệu

Thực hiện các bước làm sạch dữ liệu:

- Thay thế `"N/a"` bằng `np.nan`.
- Xác định các cột nên là số (loại trừ các cột như `Player`, `Nation`, v.v.).
- Chuyển đổi các cột số sang kiểu số, ép lỗi thành `NaN`.
- Lọc các cột hoàn toàn là số và điền `NaN` bằng 0.

3. Phân loại chỉ số tấn công và phòng thủ

Phân loại các cột số thành chỉ số tấn công và phòng thủ:

- Chỉ số tấn công: Các cột chứa từ khóa như `Goals`, `Assists`, `xG`.
- Chỉ số phòng thủ: Các cột chứa từ khóa như `Tackles`, `Interceptions`.
- Chọn ngẫu nhiên tối đa 3 cột từ mỗi loại để phân tích.

4. Xác định top 3 cao nhất và thấp nhất

Tìm top 3 cầu thủ có giá trị cao nhất và thấp nhất cho mỗi chỉ số số:

- Sắp xếp dữ liệu theo từng cột số.
- Ghi kết quả vào tệp `top_3.txt`.

5. Tính thống kê cho tất cả cầu thủ và theo đội

Tính trung vị, trung bình, và độ lệch chuẩn:

- Tính cho tất cả cầu thủ và lưu vào `all_stats`.
- Tính theo từng đội (`stats_standard_Squad`) và lưu vào `team_stats`.
- Làm phẳng MultiIndex, đổi tên cột, và gộp dữ liệu vào `results2.csv`.

6. Xác định đội có điểm số cao nhất cho mỗi chỉ số

Tính tổng các chỉ số theo đội và xác định đội có giá trị cao nhất cho mỗi chỉ số.

stats_standard_Matches Played: Southampton
stats_standard_Minutes: Leicester City
stats_standard_Goals: Liverpool
stats_standard_Assists: Liverpool
stats_standard_Yellow Cards: Chelsea
stats_standard_Red Cards: Arsenal
stats_standard_xG (Expected Goals): Liverpool
stats_standard_xAG (Exp. Assisted Goals): Liverpool
stats_standard_Progressive Carries: Manchester City

stats_standard_Progressive Passes: Liverpool
stats_standard_Progressive Passes Rec: Liverpool
stats_standard_Goals/90: Manchester City
stats_standard_Assists/90: Chelsea
stats_standard_xG/90: Aston Villa
stats_standard_xAG/90: Chelsea
stats_shooting_Shots on Target %: Tottenham
stats_shooting_Shots on target/90: Everton
stats_shooting_Goals/Shot: Manchester City
stats_shooting_Average Shot Distance: Southampton
stats_passing_Passes Completed: Manchester City
stats_passing_Pass Completion %: Southampton
stats_passing_Total Passing Distance: Manchester City
stats_passing_Passes Completed (Short): Manchester City
stats_passing_Pass Completion % (Medium): Southampton
stats_passing_Pass Completion % (Long): Manchester Utd
stats_passing_Key Passes: Liverpool
stats_passing_Passes into Final Third: Manchester City
stats_passing_Passes into Penalty Area: Arsenal
stats_passing_Crosses into Penalty Area: Bournemouth
stats_passing_Progressive Passes: Liverpool
stats_gca_Shot-Creating Actions: Liverpool
stats_gca_Shot-Creating Actions/90: Brighton
stats_gca_Goal-Creating Actions: Liverpool
stats_gca_Goal-Creating Actions/90: Brighton
stats_defense_Tackles: Manchester Utd
stats_defense_Tackles Won: Manchester Utd
stats_defense_Dribbles Challenged: Manchester Utd
stats_defense_Challenges Lost: Brighton
stats_defense_Blocks: Leicester City
stats_defense_Shots Blocked: Brentford
stats_defense_Passes Blocked: Crystal Palace
stats_defense_Interceptions: Manchester Utd
stats_possession_Touches: Manchester City
stats_possession_Touches (Def Pen): Brentford
stats_possession_Touches (Def 3rd): Manchester Utd
stats_possession_Touches (Mid 3rd): Manchester City
stats_possession_Touches (Att 3rd): Manchester City
stats_possession_Touches (Att Pen): Manchester City
stats_possession_Take-Ons Attempted: Tottenham
stats_possession_Successful Take-On %: Aston Villa
stats_possession_Tackled During Take-On %: Manchester Utd
stats_possession_Carries: Manchester City
stats_possession_Progressive Carrying Distance: Manchester City
stats_possession_Progressive Carries: Manchester City
stats_possession_Carries into Final Third: Manchester City
stats_possession_Carries into Penalty Area: Manchester City
stats_possession_Miscontrols: Bournemouth

stats_possession_Dispossessed: Newcastle Utd
stats_possession_Passes Received: Manchester City
stats_possession_Progressive Passes Rec: Liverpool
stats_misc_Yellow Cards: Chelsea
stats_misc_Red Cards: Arsenal
stats_misc_Fouls Committed: Bournemouth
stats_misc_Fouls Drawn: Tottenham
stats_misc_Offsides: Nott'ham Forest
stats_misc_Crosses: Fulham
stats_misc_Ball Recoveries: Bournemouth
stats_misc_Aerials Won: Brentford
stats_misc_Aerials Lost: Crystal Palace
stats_misc_% of Aerials Won: Southampton
stats_keeper_Goals Against/90: Leicester City
stats_keeper_Save Percentage: Bournemouth
stats_keeper_Clean Sheet Percentage: Brentford
stats_keeper_Save% (Penalty Kicks): Everton

7. Đánh giá đội thể hiện tốt nhất

Dựa trên các chỉ số chính (`Goals`, `Assists`, `xG`, `xAG`, `Save Percentage`):

- Tính trung bình các chỉ số theo đội.
- Xác định đội có tổng điểm trung bình cao nhất.

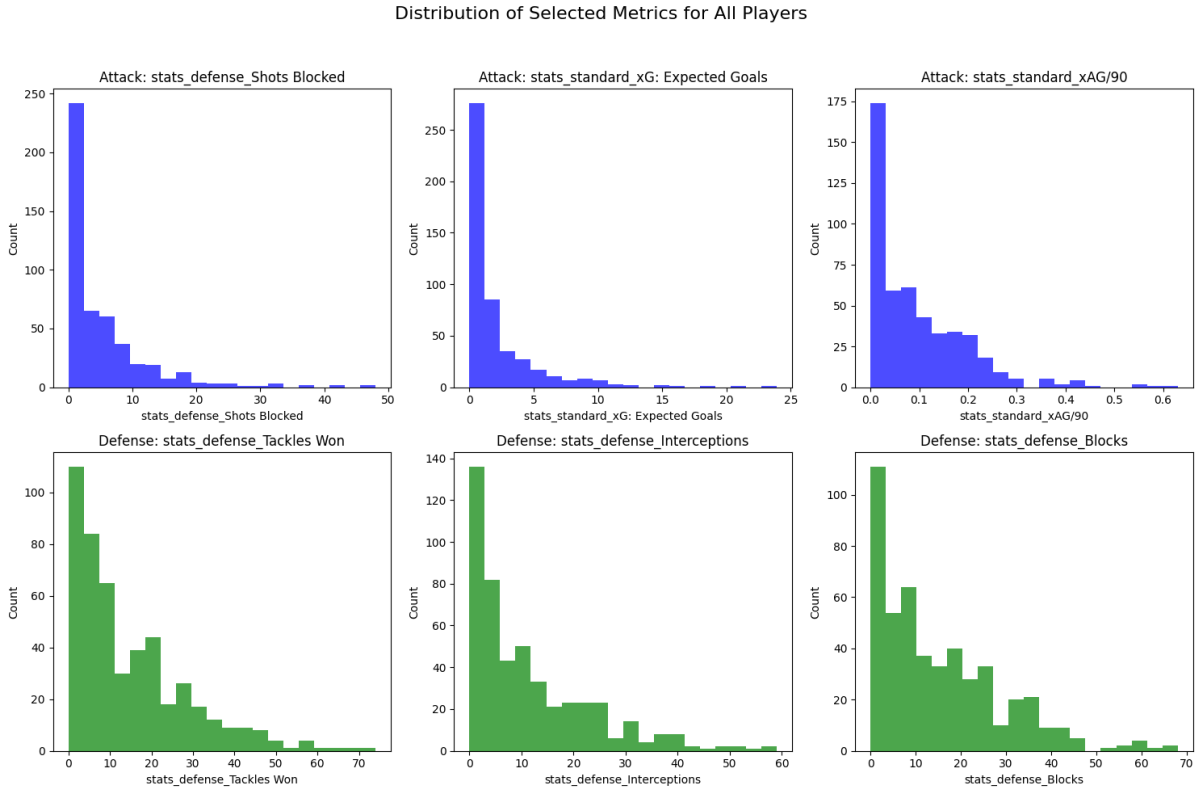
Best Performing Team: Liverpool

Reason: Liverpool leads with a total average score of **19.47** across key metrics including *goals*, *assists*, *expected goals (xG)*, *expected assisted goals (xAG)*, and *save percentage (Save%)*. This demonstrates Liverpool's consistent excellence in both offensive and defensive performance.

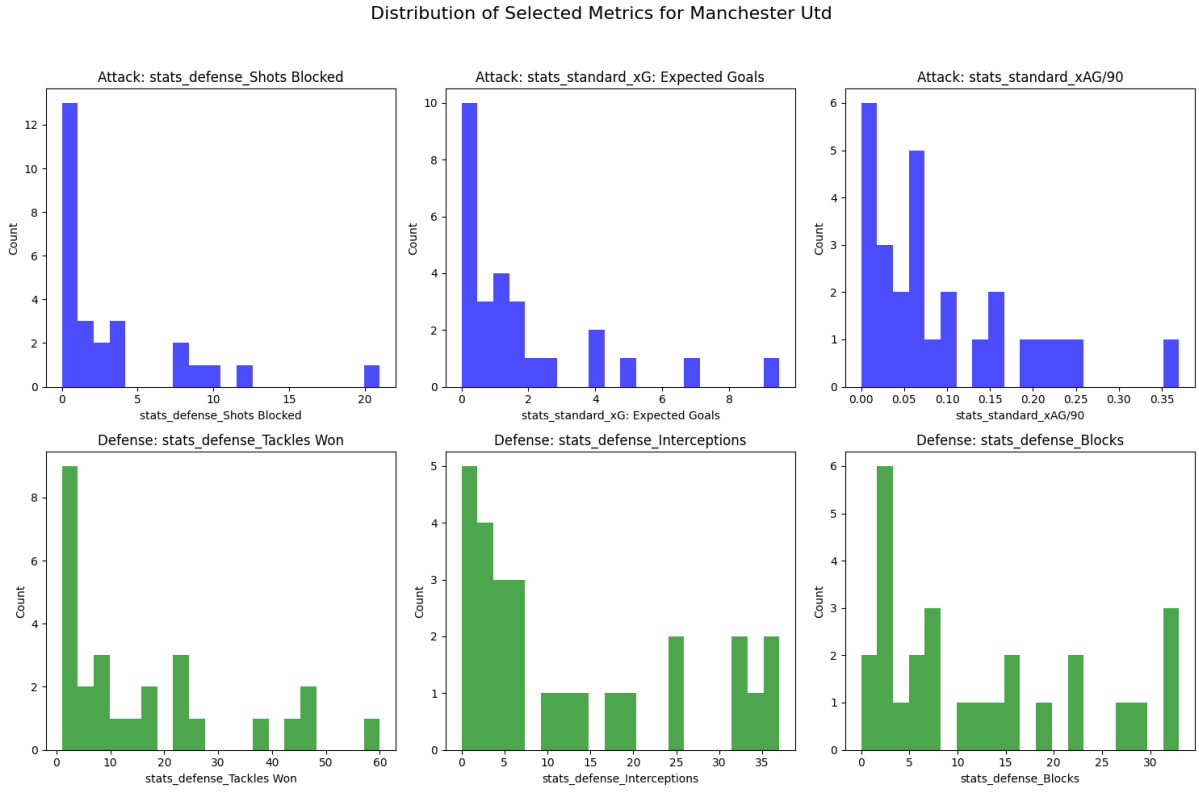
8. Vẽ biểu đồ histogram

Vẽ biểu đồ histogram cho các chỉ số tấn công và phòng thủ:

- Tạo thư mục `histograms` nếu chưa tồn tại.
- Hàm `sanitize_filename` làm sạch tên tệp.
- Vẽ histogram cho tất cả cầu thủ và lưu vào `histograms/hist_all_players.png`.
- Vẽ histogram cho từng đội và lưu vào các tệp riêng (ví dụ: `histograms/hist_[team].png`).



Hình 1: Histogram cho tất cả cầu thủ



Hình 2: Histogram cho cầu thủ của Manchester Utd

Bài 3:

Mã nguồn đầy đủ

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

# Load data from 'results.csv' file
data = pd.read_csv('results.csv')

# Select features for clustering
features = [
    'stats_passing_Passes Completed',  # Passing control
    'stats_standard_Goals',            # Number of goals
    'stats_defense_Tackles',           # Tackling ability
    'stats_standard_Assists',          # Number of assists
    'stats_shooting_Shots on target/90' # Shots on target per 90 minutes
]

# Extract features and handle missing values
data_features = data[features].fillna(0)

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_features)

# Elbow method to determine optimal number of clusters
inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow curve
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.savefig('elbow_plot.png')

```

```
# Assume optimal number of clusters is 4 (to be confirmed from the plot)
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(data_scaled)

# Reduce dimensions using PCA to 2D
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)

# Create DataFrame for PCA data
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2'])
df_pca['Cluster'] = clusters

# Plot 2D clustering visualization
plt.subplot(1, 2, 2)
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df_pca, palette='viridis', s=100)
plt.title('2D Clustering Visualization')
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
plt.savefig('cluster_plot.png')

# Save the combined plots
plt.tight_layout()
plt.savefig('combined_plots.png')
```

Phân tích chi tiết từng bước

Bước 1: Nhập các thư viện cần thiết

Đoạn mã bắt đầu bằng việc nhập các thư viện:

- `pandas` và `numpy`: Dùng để xử lý và tính toán dữ liệu.
- `sklearn.cluster.KMeans`: Thư viện cung cấp thuật toán K-Means để phân cụm.
- `sklearn.preprocessing.StandardScaler`: Chuẩn hóa dữ liệu để đảm bảo các đặc trưng có cùng thang đo.
- `sklearn.decomposition.PCA`: Giảm chiều dữ liệu để trực quan hóa.
- `matplotlib.pyplot` và `seaborn`: Dùng để vẽ biểu đồ.

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

Bước 2: Đọc dữ liệu từ tệp

Dữ liệu được đọc từ tệp `results.csv` bằng `pd.read_csv()`. Tệp này được giả định chứa dữ liệu thống kê hiệu suất của các cầu thủ bóng đá.

```
# Load data from 'results.csv' file
data = pd.read_csv('results.csv')
```

Bước 3: Chọn các đặc trưng để phân cụm

Mã chọn 5 đặc trưng từ tập dữ liệu để phân cụm:

- `stats_passing_Passes Completed`: Số đường chuyền hoàn thành (kiểm soát bóng).
- `stats_standard_Goals`: Số bàn thắng.
- `stats_defense_Tackles`: Số lần tắc bóng.
- `stats_standard_Assists`: Số đường kiến tạo.
- `stats_shooting_Shots on target/90`: Số cú sút trúng đích mỗi 90 phút.

Các đặc trưng này được lưu vào danh sách `features`, sau đó dữ liệu tương ứng được trích xuất và các giá trị thiếu được thay bằng 0 bằng `fillna(0)`.

```
# Select features for clustering
features = [
    'stats_passing_Passes Completed',  # Passing control
    'stats_standard_Goals',            # Number of goals
    'stats_defense_Tackles',           # Tackling ability
    'stats_standard_Assists',          # Number of assists
    'stats_shooting_Shots on target/90' # Shots on target per 90
       minutes
]

# Extract features and handle missing values
data_features = data[features].fillna(0)
```

Bước 4: Chuẩn hóa dữ liệu

Dữ liệu được chuẩn hóa bằng `StandardScaler` để đưa các đặc trưng về cùng thang đo (trung bình = 0, độ lệch chuẩn = 1). Điều này rất quan trọng vì K-Means dựa trên khoảng cách Euclidean, và các đặc trưng có thang đo khác nhau có thể làm sai lệch kết quả.

```
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_features)
```

Bước 5: Xác định số cụm tối ưu bằng phương pháp Elbow

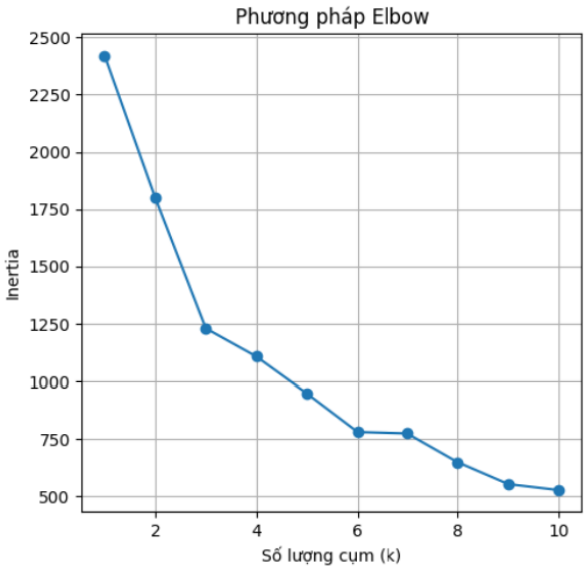
Phương pháp Elbow được sử dụng để xác định số lượng cụm (`k`) tối ưu. Mã lặp qua các giá trị `k` từ 1 đến 10, huấn luyện mô hình K-Means cho từng `k`, và tính giá trị `inertia` (tổng bình phương khoảng cách từ mỗi điểm đến tâm cụm gần nhất). Giá trị `inertia` được lưu vào danh sách để vẽ biểu đồ.

```
# Elbow method to determine optimal number of clusters
inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)
```

Bước 6: Vẽ biểu đồ Elbow

Biểu đồ Elbow được vẽ để trực quan hóa `inertia` theo `k`. Điểm "khuỷu tay" (nơi `inertia` giảm chậm lại) sẽ chỉ ra số cụm tối ưu. Biểu đồ được lưu vào tệp `elbow_plot.png`.

```
# Plot the Elbow curve
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.savefig('elbow_plot.png')
```



Hình 3: Biểu đồ Elbow

Bước 7: Áp dụng K-Means với số cụm tối ưu

Giả sử số cụm tối ưu là 4 (dựa trên biểu đồ Elbow), mã khởi tạo và huấn luyện mô hình K-Means với `k=4`, sau đó dự đoán nhãn cụm cho từng điểm dữ liệu.

```
# Assume optimal number of clusters is 4 (to be confirmed from
    the plot)
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(data_scaled)
```

Bước 8: Giảm chiều dữ liệu bằng PCA

Để trực quan hóa dữ liệu trong không gian 2 chiều, mã sử dụng PCA để giảm chiều dữ liệu từ 5 chiều (5 đặc trưng) xuống 2 chiều (PC1 và PC2).

```
# Reduce dimensions using PCA to 2D
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
```

Bước 9: Tạo DataFrame cho dữ liệu PCA

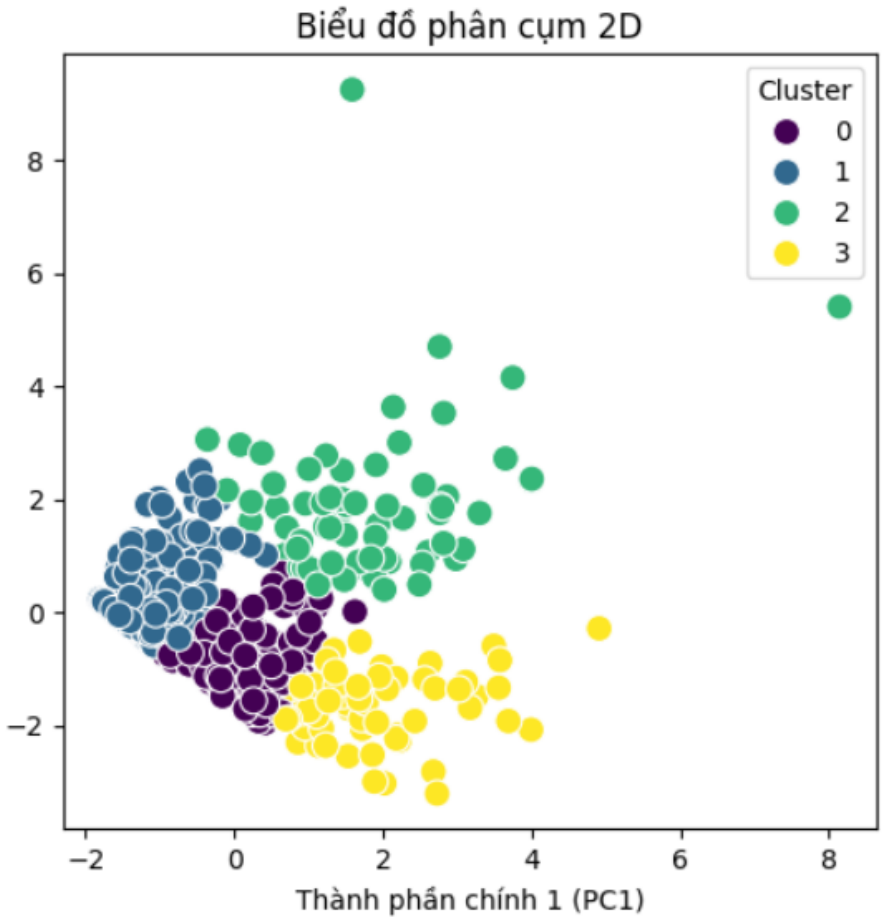
Dữ liệu sau khi giảm chiều được lưu vào một DataFrame với các cột `PC1`, `PC2`, và thêm cột `Cluster` chứa nhãn cụm.

```
# Create DataFrame for PCA data
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2'])
df_pca['Cluster'] = clusters
```

Bước 10: Vẽ biểu đồ phân cụm 2D

Biểu đồ phân tán được vẽ bằng `seaborn.scatterplot`, trong đó trục x là `PC1`, trục y là `PC2`, và màu sắc biểu thị các cụm khác nhau. Biểu đồ được lưu vào `cluster_plot.png`.

```
# Plot 2D clustering visualization
plt.subplot(1, 2, 2)
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df_pca,
    palette='viridis', s=100)
plt.title('2D Clustering Visualization')
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
plt.savefig('cluster_plot.png')
```



Hình 4: Biểu đồ phân cụm 2D

Bài 4:

Mã nguồn lấy data cho chuyển nhượng cầu thủ

```
# Function to scrape player data from a given URL
def scrape_page(driver, url):
    driver.get(url)  # Navigate to the URL
    time.sleep(3)  # Wait 3 seconds for page to load
    soup = BeautifulSoup(driver.page_source, 'html.parser')  # Parse HTML content
    # Find the table containing player data
    table = soup.find('table', class_='table table-hover no-cursor table-striped leaguetable mvp-table similar-players-table mb-0')
    if not table:  # Return empty list if table is not found
        return []

    data = []  # List to store player data
    rows = table.find('tbody').find_all('tr')  # Get all rows in table body

```

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

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

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

            # Extract estimated transfer value (ETV)
            etv_span = row.find('span', class_='player-tag')
            etv = etv_span.text.strip() if etv_span else None

            # Append data as a dictionary
            data.append({
                'player_name': player_name,
                'team': team,
                'price': etv,
                'skill/pot': skill_pot
            })
        except:  # Skip row if any error occurs
            continue

    return data  # Return list of player data


[title={Scrape data from all pages}]
# Define base URL and number of pages to scrape
base_url =
   "https://www.footballtransfers.com/en/players/uk-premier-league"
total_pages = 22
all_data = []  # List to store data from all pages

driver = setup_driver()  # Initialize Chrome driver

try:
    # Iterate through all pages
    for page in range(1, total_pages + 1):
```

```
        if page == 1:
            url = base_url  # Use base URL for first page
        else:
            url = f"{base_url}/{page}"  # Append page number for
                subsequent pages
        page_data = scrape_page(driver, url)  # Scrape data from
            current page
        all_data.extend(page_data)  # Add page data to main list
finally:
    driver.quit()  # Ensure driver is closed
    # Convert data to DataFrame and save to CSV
df_final = pd.DataFrame(all_data)  # Create DataFrame from
    collected data
df_final.to_csv('football_transfers_players.csv', index=False)  #
    Save to CSV without index
```

Mã nguồn chính

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

# Read data from CSV files
results_df = pd.read_csv('results.csv')
transfer_df = pd.read_csv('football_transfers_players.csv')

# Filter players with more than 900 minutes played
results_df = results_df[results_df['stats_standard_Minutes'] > 900]

# Standardize player names for merging
results_df['Player'] = results_df['Player'].str.lower().str.strip()
transfer_df['player_name'] = transfer_df['player_name'].str.lower().str.strip()

# Merge the two DataFrames based on player names, keeping only matches
merged_df = pd.merge(results_df, transfer_df, left_on='Player', right_on='player_name', how='inner')

# Drop rows with missing transfer prices
merged_df = merged_df.dropna(subset=['price'])

# Save the merged DataFrame to a CSV file
merged_df.to_csv('merged_results_transfer.csv', index=False, encoding='utf-8-sig')

# Function to convert transfer price strings to floats
def convert_price(price_str):
    if pd.isna(price_str) or not isinstance(price_str, str):
        return np.nan
    price_str = price_str.replace('  ', '').strip()
    if 'M' in price_str:
        return float(price_str.replace('M', ''))
    elif 'K' in price_str:
        return float(price_str.replace('K', '')) / 1000
    return np.nan

# Apply the conversion function to the 'price' column
merged_df['price'] = merged_df['price'].apply(convert_price)

```

```
# Select feature columns (stats_ columns excluding certain ones)
feature_cols = [col for col in results_df.columns if
    col.startswith('stats_') and col not in [
    'stats_standard_Nation', 'stats_standard_Squad',
        'stats_standard_Position'
]]

# Handle the age column: extract the numeric age
def extract_age(age_str):
    if pd.isna(age_str) or not isinstance(age_str, str):
        return np.nan
    try:
        return float(age_str.split('-')[0])
    except:
        return np.nan

if 'stats_standard_Current age' in feature_cols:
    merged_df['stats_standard_Current age'] =
        merged_df['stats_standard_Current age'].apply(extract_age)

# Extract features
features = merged_df[feature_cols]

# Replace 'N/a' with NaN and convert to numeric
features = features.replace('N/a', np.nan)
features = features.apply(pd.to_numeric, errors='coerce')
features = features.fillna(0)

# Target variable is the converted price
target = merged_df['price']

# Scale the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split data into training and testing sets
X_train, X_test, y_train, y_test =
    train_test_split(features_scaled, target, test_size=0.2,
    random_state=42)

# Train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100,
    random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Calculate loss (Mean Squared Error) and accuracy (R-squared)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
89
90  # Print results
91  print(f"Mean Squared Error (MSE): {mse:.2f}")
92  print(f"R-squared Score (R2): {r2:.2f}")
93
94  # Plotting results
95  plt.figure(figsize=(12, 5))
96
97  # Plot 1: Actual vs Predicted values
98  plt.subplot(1, 2, 1)
99  plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
100 plt.plot([y_test.min(), y_test.max()], [y_test.min(),
        y_test.max()], 'r--', lw=2)
101 plt.xlabel('Actual Price ( M )')
102 plt.ylabel('Predicted Price ( M )')
103 plt.title('Actual vs Predicted Prices')
104 plt.grid(True)
105
106 # Plot 2: Error distribution
107 plt.subplot(1, 2, 2)
108 errors = y_pred - y_test
109 sns.histplot(errors, kde=True, color='purple')
110 plt.xlabel('Prediction Error ( M )')
111 plt.ylabel('Frequency')
112 plt.title('Prediction Error Distribution')
113 plt.grid(True)
114
115 plt.tight_layout()
116 plt.savefig('prediction_results.png')
117 plt.close()
118
119 # Save predictions to a CSV file
120 predictions_df = pd.DataFrame({
121     'Player': merged_df.loc[y_test.index, 'Player'],
122     'Actual_Price': y_test,
123     'Predicted_Price': y_pred
124 })
125 predictions_df.to_csv('price_predictions.csv', index=False,
        encoding='utf-8-sig')
```

Phân tích chi tiết từng bước

Bước 1: Nhập các thư viện cần thiết

Đoạn mã bắt đầu bằng việc nhập các thư viện cần thiết để xử lý dữ liệu, huấn luyện mô hình, và trực quan hóa kết quả:

- `pandas` và `numpy`: Thư viện xử lý và tính toán dữ liệu.
- `sklearn.ensemble.RandomForestRegressor`: Mô hình Random Forest cho bài toán hồi quy.

- `sklearn.model_selection.train_test_split`: Chia dữ liệu thành tập huấn luyện và kiểm tra.
- `sklearn.preprocessing.StandardScaler`: Chuẩn hóa dữ liệu số.
- `sklearn.metrics`: Các hàm tính toán độ lỗi (MSE) và độ chính xác (R-squared).
- `matplotlib.pyplot` và `seaborn`: Trực quan hóa dữ liệu qua biểu đồ.

Bước 2: Đọc dữ liệu từ tệp CSV

Dữ liệu được đọc từ hai tệp CSV: `results.csv` chứa thông tin thống kê cầu thủ và `football_transfers_players.csv` chứa giá trị chuyển nhượng.

Bước 3: Lọc cầu thủ có thời gian thi đấu trên 900 phút

Lọc dữ liệu để chỉ giữ lại các cầu thủ có thời gian thi đấu (`stats_standard_Minutes`) lớn hơn 900 phút, đảm bảo dữ liệu đủ ý nghĩa để phân tích.

Bước 4: Chuẩn hóa tên cầu thủ

Tên cầu thủ trong cả hai bảng được chuyển thành chữ thường và loại bỏ khoảng trắng thừa, giúp việc hợp nhất dữ liệu chính xác hơn.

Bước 5: Hợp nhất hai DataFrame

Hai bảng dữ liệu được hợp nhất dựa trên tên cầu thủ bằng phép nối trong (`inner join`), chỉ giữ lại các cầu thủ xuất hiện ở cả hai tập dữ liệu.

Bước 6: Loại bỏ các hàng thiếu giá trị chuyển nhượng

Các hàng không có giá trị trong cột `price` bị loại bỏ để đảm bảo mô hình hoạt động với dữ liệu đầy đủ.

Bước 7: Lưu DataFrame hợp nhất

Dữ liệu sau khi hợp nhất được lưu vào tệp `merged_results_transfer.csv` với mã hóa `utf-8-sig` để hỗ trợ ký tự tiếng Việt.

Bước 8: Chuyển đổi giá trị chuyển nhượng từ chuỗi sang số

Hàm `convert_price` xử lý chuỗi giá trị như '€50M' hoặc '€500K', loại bỏ ký tự '€', chuyển 'M' thành triệu và 'K' thành nghìn, trả về số thực.

Bước 9: Chọn các cột đặc trưng

Các cột bắt đầu bằng `stats_` được chọn làm đặc trưng, ngoại trừ các cột không phải số như quốc gia, đội bóng, và vị trí.

Bước 10: Xử lý cột tuổi

Hàm `extract_age` tách tuổi từ chuỗi như '26-352', chỉ lấy phần số năm (26) để sử dụng làm đặc trưng.

Bước 11: Trích xuất đặc trưng

Các cột đặc trưng được trích xuất từ DataFrame hợp nhất để chuẩn bị cho việc huấn luyện mô hình.

Bước 12: Thay thế 'N/a' và chuyển sang kiểu số

Giá trị 'N/a' được thay bằng `NaN`, các cột được chuyển thành kiểu số, và các giá trị `NaN` còn lại được điền bằng 0.

Bước 13: Xác định biến mục tiêu

Biến mục tiêu là cột `price` đã được chuyển đổi sang dạng số thực.

Bước 14: Chuẩn hóa đặc trưng

Sử dụng `StandardScaler` để chuẩn hóa các đặc trưng, giúp mô hình học tốt hơn trên dữ liệu có thang đo khác nhau.

Bước 15: Chia dữ liệu thành tập huấn luyện và kiểm tra

Dữ liệu được chia thành tập huấn luyện (80%) và tập kiểm tra (20%) với `random_state=42` để đảm bảo kết quả tái lập.

Bước 16: Huấn luyện mô hình Random Forest

Mô hình Random Forest với 100 cây quyết định được khởi tạo và huấn luyện trên tập huấn luyện.

Bước 17: Dự đoán trên tập kiểm tra

Mô hình dự đoán giá trị chuyển nhượng trên tập kiểm tra, kết quả được lưu vào `y_pred`.

Bước 18: Tính toán hàm loss và độ chính xác

Tính toán Mean Squared Error (MSE) để đo lường độ lỗi và R-squared (R2) để đánh giá độ chính xác của mô hình.

Bước 19: Vẽ biểu đồ kết quả

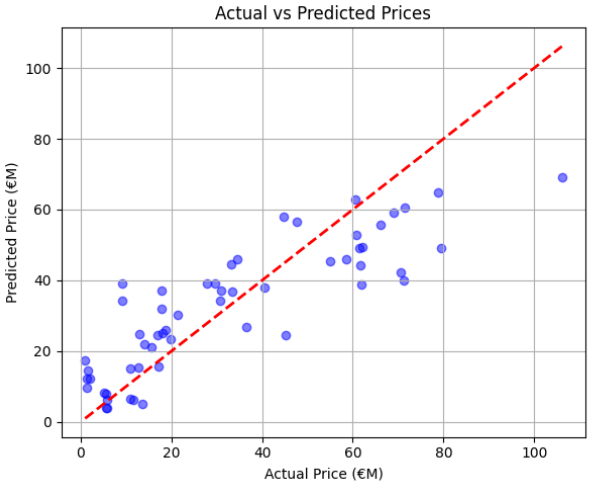
Hai biểu đồ được tạo ra:

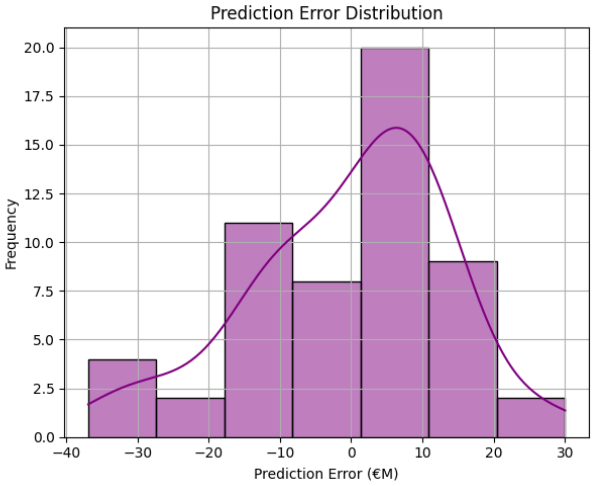
- Biểu đồ phân tán so sánh giá trị thực tế và dự đoán, với đường tham chiếu màu đỏ.
- Biểu đồ phân phối sai số dự đoán, sử dụng histogram và đường cong mật độ.

Kết quả được lưu vào tệp `prediction_results.png`.

Bước 20: Lưu kết quả dự đoán

Dữ liệu dự đoán (tên cầu thủ, giá trị thực tế, giá trị dự đoán) được lưu vào tệp `price_predictions.cs` với mã hóa `utf-8-sig`.


Actual vs Predicted Prices
Predicted Price (€M)
Actual Price (€M)
0
20
40
60
80
100


Prediction Error Distribution
Frequency
Prediction Error (€M)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−40
−30
−20
−10
0
10
20
30