HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

PYTHON PROGRAM REPORT

Giảng viên hướng dẫn	**: Kim Ngọc Bách**
Họ và tên sinh viên	**: Nguyễn Đức Mạnh**
Mã sinh viên	**: B23DCDT169**
Lớp	**: D23CQCE04-B**

Hà Nội – 2025

TABLE OF CONTENT

PART I :

Write a Python program to collect footballer player statistical data with the following requirements:

• Collect statistical data [*] for all players who have played more than 90 minutes in the 2024-2025 English Premier League season.

• Data source: https://fbref.com/en/

• Save the result to a file named 'results.csv', where the result table has the following structure:

o Each column corresponds to a statistic.

o Players are sorted alphabetically by their first name.

o Any statistic that is unavailable or inapplicable should be marked as "N/a".

• [*] The required statistics are:

o Nation

o Team

o Position

o Age

o Playing Time: matches played, starts, minutes

o Performance: goals, assists, yellow cards, red cards

o Expected: expected goals (xG), expedted Assist Goals (xAG)

o Progression: PrgC, PrgP, PrgR

o Per 90 minutes: Gls, Ast, xG, xGA

o Goalkeeping:

▪ 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%

o Reference: https://fbref.com/en/squads/822bd0ba/Liverpool-Stats

I. Idea for the assignment:

- **Web Scraping**: Uses Selenium and BeautifulSoup to scrape tables containing player statistics from multiple pages on FBref.com.
- **Data Storage**: Saves each scraped table as a separate CSV file.
- **Data Cleaning**: Removes redundant header rows from the CSV files.
- **Data Merging**: Combines relevant columns from multiple CSV files into a single dataset, filtering for players with more than 90 minutes played.
- **Data Transformation**: Standardizes column names, extracts first names, sorts data, and fills missing values.
- **Output**: Saves the final processed dataset to a CSV file (results.csv).

II. Code Analysis:

1. Imports:

```
import random
import time
import pandas as pd
from bs4 import BeautifulSoup, Comment
from io import StringIO
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
```

- **Purpose**: Imports necessary libraries for web scraping, data manipulation, and browser automation.
- **Key Libraries:**
  - random and time: Used for adding random delays to avoid detection as a bot.
  - pandas: For reading, manipulating, and saving CSV files.
  - BeautifulSoup and Comment: For parsing HTML and handling commented-out tables.
  - io.StringIO: To convert HTML table strings into a format pandas can read.
  - selenium: For automating a Chrome browser to load dynamic content.
  - webdriver_manager: Automatically manages the ChromeDriver installation.

2. Browser Setup

```
options = Options()
options.add_argument("--disable-blink-features=AutomationControlled")
options.add_experimental_option("excludeSwitches", ["enable-automation"])
options.add_experimental_option('useAutomationExtension', False)
options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
                     "AppleWebKit/537.36 (KHTML, like Gecko) "
                     "Chrome/122.0.0.0 Safari/537.36")

browser = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
browser.execute_cdp_cmd("Page.addScriptToEvaluateOnNewDocument", {
    "source": """
        Object.defineProperty(navigator, 'webdriver', {get: () => undefined})
        """
})
```

- **Purpose**: Configures a Chrome browser instance to avoid detection as an automated bot and mimic a real user.
- **Key Configurations**:

- --disable-blink-features=AutomationControlled: Disables browser features that signal automation.
- excludeSwitches and useAutomationExtension: Removes automation-related flags.
- user-agent: Sets a realistic user agent string to emulate a typical Chrome browser on Windows.
- ChromeDriverManager().install(): Automatically downloads and sets up the appropriate ChromeDriver.
- execute_cdp_cmd: Executes a Chrome DevTools Protocol command to hide the navigator.webdriver property, further masking automation.

3. List of Tables to Scrape

```
stat_sources = [
    ("https://fbref.com/en/comps/9/stats/Premier-League-Stats#all_stats_standard", "all_stats_standard", "D:/btl/player_stats_standard.csv"),
    ("https://fbref.com/en/comps/9/keepers/Premier-League-Stats#all_stats_keeper_saves", "all_stats_keeper", "D:/btl/player_goalkeeping.csv"),
    ("https://fbref.com/en/comps/9/shooting/Premier-League-Stats#all_stats_shooting", "all_stats_shooting", "D:/btl/player_shooting.csv"),
    ("https://fbref.com/en/comps/9/passing/Premier-League-Stats#all_stats_passing", "all_stats_passing", "D:/btl/player_passing.csv"),
    ("https://fbref.com/en/comps/9/gca/Premier-League-Stats#all_stats_gca", "all_stats_gca", "D:/btl/player_gca.csv"),
    ("https://fbref.com/en/comps/9/defense/Premier-League-Stats#all_stats_defense", "all_stats_defense", "D:/btl/player_defense.csv"),
    ("https://fbref.com/en/comps/9/possession/Premier-League-Stats#all_stats_possession", "all_stats_possession", "D:/btl/player_possession.csv"),
    ("https://fbref.com/en/comps/9/misc/Premier-League-Stats#all_stats_misc", "all_stats_misc", "D:/btl/player_misc.csv")
]
```

➢ **Purpose**: Defines a list of tuples, each containing:

- The URL of the page to scrape.
- The HTML div ID containing the target table.
- The file path where the scraped table will be saved as a CSV.

➢ **Details**:

- Covers eight categories of statistics: standard, goalkeeping, shooting, passing, goal creation (gca), defense, possession, and miscellaneous.
- File paths are hardcoded to D:/btl/, which assumes a specific directory structure on the user's machine.

4. Scraping Loop

```
for url, div_id, file_path in stat_sources:
        browser.get(url)
        time.sleep(random.uniform(3.5, 7.5))
        soup = BeautifulSoup(browser.page_source, "html.parser")
        div = soup.find("div", id=div_id)
        comment = div.find(string=lambda text: isinstance(text, Comment)) if div else None
        table_html = BeautifulSoup(comment, "html.parser").find("table") if comment else div.find("table") if div else None
        if table_html:
            df = pd.read_html(StringIO(str(table_html)))[0]
            df.to_csv(file_path, index=False, encoding="utf-8-sig")
            print(f"\u2714\ufe0f Lưu: {file_path}")
        else:
            print(f"\u274c Không tìm thấy bảng: {file_path}")

browser.quit()
```

➢ **Purpose**: Iterates through the stat_sources list to scrape each table and save it as a CSV.

➢ **Step-by-Step**:

1. **Navigate to URL**: browser.get(url) loads the page.
2. **Random Delay**: time.sleep(random.uniform(3.5, 7.5)) adds a random delay (3.5–7.5 seconds) to mimic human behavior and avoid rate-limiting or bans.
3. **Parse HTML**: BeautifulSoup(browser.page_source, "html.parser") creates a BeautifulSoup object from the page's HTML.
4. **Find Table**:

- Locates the div with the specified div_id.
- Checks for a commented-out table (common on FBref, where tables are stored in HTML comments to reduce initial page load).
- If a comment is found, parses it to extract the table; otherwise, looks for a direct <table> element.

5. **Convert to DataFrame**: Uses pd.read_html to convert the HTML table into a pandas DataFrame.
6. **Save to CSV**: Saves the DataFrame to the specified file_path with UTF-8 encoding (utf-8-sig to handle special characters, e.g., player names with accents).
7. **Feedback**: Prints a success (✔) or failure (❌) message for each file.
8. **Cleanup**: browser.quit() closes the browser after all tables are scraped.

➢ **Notable Features**:

- Handles commented-out tables, which is specific to FBref's structure.
- Includes error handling for cases where the table or div is not found.
- Uses encoding="utf-8-sig" to ensure compatibility with special characters.

5. Remove Duplicate Headers

```
for path in [item[2] for item in stat_sources]:
        with open(path, 'r', encoding='utf-8') as f:
            lines = f.readlines()
        with open(path, 'w', encoding='utf-8') as f:
            f.writelines(lines[1:])
```

- **Purpose**: Removes the first line (assumed to be a duplicate header) from each CSV file.
- **Details**:
    - Iterates over the file paths in stat_sources.
    - Reads all lines of the CSV file.
    - Writes back all lines except the first one (lines[1:]).
    - Uses utf-8 encoding for consistency.
- **Potential Issue**: Assumes the first line is always a duplicate header without verification, which could lead to data loss if the assumption is incorrect.

6. Helper Function for Merging Data

```
def combine_stat_table(main_df, filepath, selected_cols):
        try:
            df = pd.read_csv(filepath)
            df.columns = df.columns.str.strip()
            return pd.merge(main_df, df[["Player", "Squad"] + selected_cols], on=["Player", "Squad"], how="left")
        except:
            return main_df
```

- **Purpose**: Merges selected columns from a secondary CSV file into the main DataFrame.
- **Parameters**:
    - main_df: The primary DataFrame to merge into.
    - filepath: Path to the secondary CSV file.
    - selected_cols: List of columns to include from the secondary CSV.
- **Details**:
    - Reads the secondary CSV into a DataFrame.
    - Strips whitespace from column names to ensure consistency.

- Performs a left merge on Player and Squad columns, keeping all rows from main_df.
- Returns main_df unchanged if an error occurs (e.g., file not found or incompatible data).

➢ **Notable Features**:

- Uses how="left" to preserve all players in main_df, even if they lack data in the secondary CSV.
- Error handling ensures the script doesn't crash on missing or malformed files.

7. Data Processing and Merging

```
data = pd.read_csv("D:/btl/player_stats_standard.csv")
data.columns = data.columns.str.strip()
data = data[pd.to_numeric(data["Min"], errors="coerce") > 90].copy()
data["First Name"] = data["Player"].apply(lambda x: x.split()[0])
data.sort_values("First Name", inplace=True)

columns_to_keep = ["Player", "Nation", "Squad", "Pos", "Age", "MP", "Starts", "Min",
                   "Gls", "Ast", "CrdY", "CrdR", "xG", "xAG", "PrgC", "PrgP", "PrgR",
                   "Gls.1", "Ast.1", "xG.1", "xAG.1"]
data = data[columns_to_keep]
```

➢ **Purpose**: Loads the standard stats CSV, filters and transforms it, and selects a subset of columns.

➢ **Step-by-Step**:

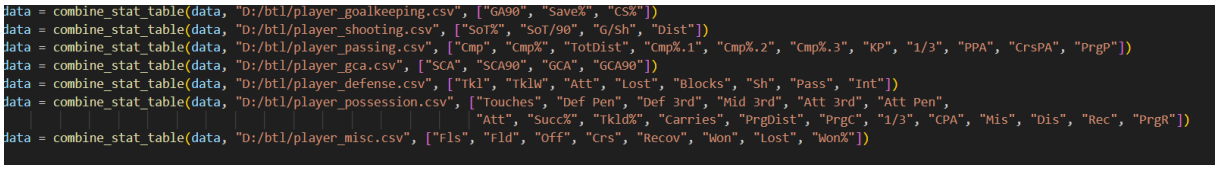
1. **Load Data**: Reads player_stats_standard.csv into a DataFrame.
2. **Clean Columns**: Strips whitespace from column names.
3. **Filter Players**: Keeps only players with more than 90 minutes played (Min column), converting non-numeric values to NaN (errors="coerce").

4. **Add First Name**: Creates a First Name column by extracting the first word from the Player column (e.g., "Kevin De Bruyne" → "Kevin").
5. **Sort**: Sorts the DataFrame by First Name.
6. **Select Columns**: Keeps only the specified columns (e.g., player info, goals, assists, expected goals, progressive carries/passes/receptions).

➢ **Notable Features**:

- The Min > 90 filter ensures only players with significant playing time are included.
- The First Name column is likely for sorting or display purposes but isn't used further in the script.
- The selected columns focus on key performance metrics, excluding less relevant stats.

8. Merge Additional Statistics



➢ **Purpose:** Combine additional statistical columns from other CSV files into the main DataFrame.

➢ **Details:**

- Call the combine_stat_table function for each sub-CSV file, specifying the columns to pull (e.g. save percentage for goalkeepers, shots on target percentage for kicks, etc.).
- Each call adds relevant columns to the data via left join.
- Notable features:

- Selective column selection ensures that only important metrics are pulled from each category.
- Left join retains all players from the main DataFrame, even if they do not have data in the sub-files (e.g. non-goalkeepers will not have data in player_goalkeeping.csv).

9. Standardize and save

```
data.fillna("N/a", inplace=True)
data.to_csv("D:/btl/results.csv", index=False, encoding="utf-8-sig")
print("\n✓\ufe0f Hoàn thành! File: D:/btl/results.csv")
```

- **Purpose:** Normalize data and save final results.
- **Details:**
- Fill missing values with "N/a" string for consistency.
- Save DataFrame to results.csv with utf-8-sig encoding.
- Print completion message with file path.
- **Notable features**:
- Using "N/a" instead of missing values makes CSV file easier to read.
- Utf-8-sig encoding ensures compatibility with special characters.

PART II:

Identify the top 3 players with the highest and lowest scores for each statistic. Save result

to a file name ‘top_3.txt’

• Find the median for each statistic. Calculate the mean and standard deviation for each

statistic across all players and for each team. Save the results to a file named 'results2.csv'

with the following format:

		Median of Atttribute 1	Mean of Atttribute 1	Std of Atttribute 1	...	...
0	all					
1	Team 1					
...						
n	Team n					

Plot a histogram showing the distribution of each statistic for all players in the league and

each team.

• Identify the team with the highest scores for each statistic. Based on your analysis, which

team do you think is performing the best in the 2024-2025 Premier League season?

I. Idea for the assignment:

- **Data Conversion**: Standardizes data columns (e.g., age, percentages, distances, numerics) to ensure they are in a numeric format suitable for analysis.
- **Top/Bottom Statistics**: Identifies the top 3 and bottom 3 players for each metric, saving results to top_3.txt.
- **Team Summaries**: Computes median, mean, and standard deviation for each metric, aggregated for all data and per team, saving to results2.csv.
- **Histogram Plotting**: Generates histograms for each metric, both for all data and per team, saving them to the histograms directory.
- **Best Team Analysis**: Determines the team with the most leading metrics, saving results to best_team_analysis.txt.

II. Code Analysis:

1. Imports

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

- **Purpose**: Imports libraries required for data processing, statistical analysis, visualization, and file operations.
- **Key Libraries**:

- *pandas:* Handles tabular data (DataFrames) for analysis.
- *numpy:* Supports handling missing values (np.nan) and numerical computations.
- *matplotlib.pyplot:* Creates histogram visualizations.
- *os:* Manages directories (e.g., creating the histograms folder).
- *re*: Uses regular expressions for cleaning file names.

2. Conversion Function

```
def convert_numeric(x):
    if pd.isna(x) or x == 'N/a': return np.nan
    try:
        return float(str(x).split('/')[0]) if '/' in str(x) else float(x)
    except: return np.nan
```

- **Purpose**: Converts a value x into a numeric format (float) while handling inconsistencies in the data.
- **Logic**:

  - If x is NaN (using pd.isna) or equals 'N/a', return np.nan.
  - If x contains a '/' (e.g., '5/10'), split the string and take the first part ('5'), then convert it to a float.
  - Otherwise, convert x directly to a float.

- If conversion fails (e.g., due to invalid data), return np.nan.

➢ **Use Case**: Handles cases where data might be in formats like '5/10' (possibly indicating a fraction or split metric) or non-numeric values, ensuring consistency for numerical analysis.

3. Load Data

```
try:
    df = pd.read_csv("results.csv")
except:
    print("Error: results.csv file not found.")
    exit()
```

➢ **Purpose**: Loads the dataset from a CSV file named results.csv into a pandas DataFrame (df).

➢ **Error Handling:**

- If the file is not found or cannot be read, the script prints an error message and exits.

➢ **Assumption:** The dataset contains columns like 'Player', 'Squad', and statistical metrics (e.g., 'Gls', 'xG').

4. Select Columns

```
selected_columns = ['Gls', 'xG', 'Sh', 'Tkl', 'Int', 'Blocks']
columns = [c for c in selected_columns if c in df.columns]
```

➢ **Purpose**: Defines a list of columns to analyze (selected_columns) and filters them to only include those present in the DataFrame.

➢ **Columns**:

- 'Gls': Goals scored.

- 'xG': Expected goals.
- 'Sh': Shots.
- 'Tkl': Tackles.
- 'Int': Interceptions.
- 'Blocks': Blocks.

➢ **Logic**: The list comprehension ensures the script only processes columns that exist in df, avoiding errors if some columns are missing.

5. Data Cleaning

```
for c in columns:
    df[c] = df[c].apply(convert_numeric)

df[columns] = df[columns].replace('N/a', np.nan)
df[columns] = df[columns].apply(pd.to_numeric, errors='coerce')
```

➢ **Purpose**: Cleans the selected columns to ensure they contain numeric data.

➢ **Steps**:

- **Apply convert_numeric**: Uses the previously defined function to convert each value in the selected columns, handling cases like '5/10' or 'N/a'.
- **Replace 'N/a' with np.nan**: Ensures consistency in representing missing data.

- **Convert to numeric**: Uses pd.to_numeric with errors='coerce' to force non-numeric values to NaN, ensuring the columns are ready for numerical analysis.

➢ **Outcome**: The selected columns are now in a clean, numeric format, with missing or invalid data represented as NaN.

6. Top/Bottom 3 Statistics

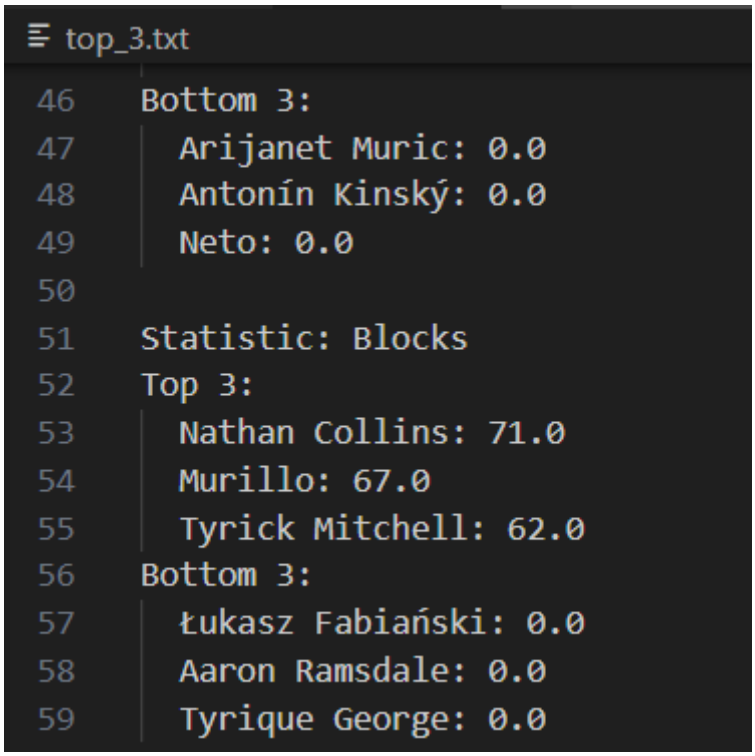
```
with open("top_3.txt", "w", encoding="utf-8") as out:
    for col in columns:
        if df[col].dropna().empty: continue
        top = df[['Player', col]].dropna().sort_values(by=col, ascending=False).head(3)
        bottom = df[['Player', col]].dropna().sort_values(by=col).head(3)
        out.write(f"Statistic: {col}\nTop 3:\n")
        for _, r in top.iterrows(): out.write(f"  {r['Player']}: {r[col]}\n")
        out.write("Bottom 3:\n")
        for _, r in bottom.iterrows(): out.write(f"  {r['Player']}: {r[col]}\n")
        out.write("\n")
```

➢ **Purpose**: Identifies the top 3 and bottom 3 players for each metric and writes the results to top_3.txt.

➢ **Logic**:

- Opens a file top_3.txt in write mode with UTF-8 encoding.
- For each column in columns:
    - Skips the column if it has no valid (non-NaN) data after dropping NaN values.
    - **Top 3**: Selects the 'Player' and current column, drops NaN values, sorts in descending order, and takes the top 3 rows.
    - **Bottom 3**: Similar to top 3, but sorts in ascending order.
    - Writes the results to the file in a formatted way (e.g., Statistic: Gls\nTop 3:\n Player1: 10\n...).

- **Outcome**: A text file (top_3.txt) is created, listing the top 3 and bottom 3 players for each metric, making it easy to identify standout performers and underperformers.



7. Summary Statistics by Team

```
summaries = []
all_stats = {'Team': 'all'}
for col in columns:
    all_stats[f'Median of {col}'] = df[col].median()
    all_stats[f'Mean of {col}'] = df[col].mean()
    all_stats[f'Std of {col}'] = df[col].std()
summaries.append(all_stats)

for team in df['Squad'].unique():
    team_data = df[df['Squad'] == team]
    row = {'Team': team}
    for col in columns:
        row[f'Median of {col}'] = team_data[col].median()
        row[f'Mean of {col}'] = team_data[col].mean()
        row[f'Std of {col}'] = team_data[col].std()
    summaries.append(row)

pd.DataFrame(summaries).to_csv("results2.csv", index=False)
```

- **Purpose**: Computes summary statistics (median, mean, standard deviation) for each metric, both overall and per team, and saves them to results2.csv.
- **Logic**:

- **Overall Statistics**:
  - Creates a dictionary all_stats with key 'Team': 'all'.
  - For each column, computes the median, mean, and standard deviation of the entire dataset and adds them to all_stats.
  - Appends all_stats to the summaries list.
- **Per-Team Statistics**:
  - Loops through each unique team in the 'Squad' column.
  - Filters the DataFrame for the current team (team_data).
  - Creates a dictionary row with the team name and computes the median, mean, and standard deviation for each metric.

- Appends row to summaries.

- **Save to CSV**:
  - Converts summaries (a list of dictionaries) into a pandas DataFrame and saves it to results2.csv without an index.

➢ **Outcome**: A CSV file (results2.csv) is created, containing summary statistics for all teams and the overall dataset, allowing for easy comparison of team performance.

results2.csv

Team	Median of Gls	Mean of Gls	Std of Gls	Median of xG	Mean of xG	Std of xG	Median of Sh	Mean of Sh	Std of Sh	Median of Tkl	Mean of Tkl	S
all	1.0	2.0303643724696356	3.548717602392066	0.9	2.0491902834008098	3.1236176418767605	2.0	5.228744939271255	7.3871704785005796	17.0	24.89676113360324	22.863
West Ham	0.0	1.48	2.6	1.2	1.776	2.254861414810232	4.0	6.28	8.228608630868283	17.0	23.6	20.816
Southampton	0.0	0.8275862068965517	1.0713464664764276	0.5	1.0724137931034483	1.3879317994062084	2.0	5.896551724137931	9.135272408113694	13.0	20.413793103448278	19.729
Everton	1.0	1.5	2.132514723892674	1.1	1.7000000000000002	2.018957769788778	3.0	5.863636363636363	9.387405212484259	26.0	30.863636363636363	28.235
Manchester City	1.0	2.64	4.414748010928823	1.3	2.5200000000000005	4.175424130153328	1.0	3.28	4.817675788178362	15.0	18.44	17.058
Leicester City	0.0	1.1153846153846154	2.006527808467614	0.4	1.1846153846153844	2.171578652997364	4.0	6.769230769230769	8.199061860031023	21.5	25.846153846153847	22.301
Bournemouth	1.0	2.347826086956522	3.5879684762496544	1.1	2.6826086956521737	3.5771563330151	3.0	5.739130434782608	7.2627154407372005	21.0	27.956521739130434	23.433
Crystal Palace	0.0	1.9047619047619047	3.4044788427123747	1.1	2.5523809523809526	3.4639022860957045	3.0	6.142857142857143	6.9085867895374545	21.0	33.142857142857146	29.935
Brighton	1.0	1.9642857142857142	2.999779533345284	0.75	1.910714285714286	2.643558108099664	2.0	4.0	5.450110430413654	16.5	22.642857142857142	19.761
Fulham	0.5	2.227272727272727	3.2649920612726087	1.05	2.1227272727272273	2.6848661965741441	2.0	4.590909090909091	6.299350615882417	23.5	28.272727272727273	23.005
Manchester Utd	0.0	1.3333333333333333	2.2641635942681337	0.7	1.6033333333333333	2.2676865521607197	1.0	3.2666666666666666	4.7845536910968525	13.0	25.1	27.822
Ipswich Town	0.0	1.1333333333333333	2.3153659624840922	0.6	1.0699999999999998	1.7597315626008965	2.0	5.766666666666667	9.626412420624519	12.5	18.4	16.581
Newcastle Utd	0.0	2.782608695652174	5.187016323467755	0.5	2.6347826086956525	4.436994139735529	3.0	5.6521739130434785	7.1896939854727553	22.0	24.434782608695652	22.022
Liverpool	1.0	3.8095238095238093	6.493219907095766	1.7	3.6761904761904765	5.513973590969107	2.0	4.190476190476191	4.718252299517775	26.0	28.476190476190474	23.932
Aston Villa	1.0	1.8928571428571428	3.2924074914790076	0.8	1.9535714285714287	2.9725263158486417	2.5	3.607142857142857	4.3148261688884484	10.5	21.392857142857142	20.552
Wolves	1.0	2.1739130434782608	3.9387604639143587	0.8	1.7608695652173914	2.4798715381748715	4.0	5.869565217391305	6.0475716225311168	21.0	32.08695652173913	28.195
Nottham Forest	1.0	2.409090909090909	4.159430946760122	1.4	1.9227272727272728	2.6047450872776343	3.5	6.818181818181818	10.064725594803928	20.5	28.727272727272727	25.941
Tottenham	1.0	2.2222222222222223	3.250246538972477	0.7	2.0999999999999996	2.9091368003373312	5.0	5.2592592592592595	5.2300359801129 81	14.0	22.85185185185185	20.776
Chelsea	1.0	2.269230769230769	3.6502897672123757	0.75	2.5653846153846156	4.209507553877751	1.5	3.730769230769231	5.632460864010986	11.0	20.730769230769 23	23.883
Brentford	0.0	2.9047619047619047	5.539898572219188	0.8	2.6095238095238096	4.196773136816519	4.0	8.714285714285714	12.942730999069928	25.0	27.523809523809526	22.234
Arsenal	2.0	2.8181818181818183	2.805375174576454	1.65	2.563636363636363	2.64098564835275	1.5	4.409090909090909	6.835893301495952	19.5	24.636363636363637	20.726

results2.csv

Median of Sh	Mean of Sh	Std of Sh	Median of Tkl	Mean of Tkl	Std of Tkl	Median of Int	Mean of Int	Std of Int	Median of Blocks	Mean of Blocks	Std of Blocks
2.0	5.228744939271255	7.387170478005796	17.0	24.89676113360324	22.86328246837779	7.0	11.323886639676113	11.865214822936 23	12.0	15.465587044534413	13.862766611805219
4.0	6.28	8.228608630868283	17.0	23.6	20.816659994661325	8.0	12.24	14.774076395272001	14.0	16.64	13.409822270758601
2.0	5.896551724137931	9.135272408113694	13.0	20.413793103448278	19.729450867347865	5.0	9.620689655172415	12.263225493620721	8.0	13.379310344827585	14.915030274150226
3.0	5.863636363636363	9.387405212484259	26.0	30.863636363636363	28.235903275196172	9.5	13.863636363636363	13.046785608670618	13.0	16.363636363636363	14.334591709585574
1.0	3.28	4.817675788178362	15.0	18.44	17.05891751157343	5.0	8.2	8.582928793055823	11.0	12.44	9.600694419328914
4.0	6.769230769230769	8.199061860031023	21.5	25.846153846153847	22.301017569056903	9.5	11.769230769230 77	11.907334520564012	16.5	17.730769230769 23	14.001593315927133
3.0	5.739130434782608	7.2627154407372005	21.0	27.956521739130434	23.433616608449036	10.0	14.391304347826088	15.622650020911603	16.0	18.826086956521 74	16.694180188733828
3.0	6.142857142857143	6.908586789537454	21.0	33.142857142857146	29.935406652133 05	11.0	14.238095238095237	14.484145683832244	19.0	21.047619047619047	18.017980437541247
2.0	4.0	5.450110430413654	16.5	22.642857142857142	19.761943010098 77	8.5	10.357142857142858	9.945087325145108	12.0	14.428571428571429	12.257812539518143
2.0	4.590909090909091	6.299350615882417	23.5	28.272727272727273	23.005551748554332	8.5	12.727272727272727	13.718884510330723	14.0	15.454545454545455	11.206831559056896
1.0	3.2666666666666666	4.7845536910968525	13.0	25.1	27.822528703341753	5.0	11.466666666666667	12.721779677424047	8.0	11.6	11.180648306414263
2.0	5.766666666666667	9.626412420624519	12.5	18.4	16.581252396441 07	6.5	9.666666666666666	9.440679889796561	8.0	13.433333333333334	13.957282942121536
3.0	5.652173913043478 5	7.189693985472753	22.0	24.434782608695652	22.022356588778 6	6.0	11.043478260869565	11.845514152353068	17.0	17.130434782608695	14.194443047852747
2.0	4.190476190476191	4.718252299517775	26.0	28.476190476190474	23.932444604801756	7.0	13.095238095238095	15.952757635922266	9.0	15.952380952380953	14.796878692738515
2.5	3.607142857142857	4.314826168888484	10.5	21.392857142857142	20.552262713518 98	5.0	8.071428571428571	8.379554356702464	10.0	10.857142857142858	9.151863949852716
4.0	5.869565217391305	6.047571622531168	21.0	32.08695652173913	28.195604176212647	9.0	12.08695652173913	9.82997754310882	16.0	18.043478260869566	12.607639317888202
3.5	6.818181818181818	10.064725594803928	20.5	28.727272727272727	25.941659487582843	11.0	13.5	11.342083456881154	11.0	18.272727272727273	18.03219823427452
5.0	5.259259259259595	5.230035980112981	14.0	22.851851851851 85	20.776657923636 5	10.0	10.777777777777779	10.82494744465243	15.0	15.148148148148149	12.249658405609972
1.5	3.730769230769231	5.632460864010986	11.0	20.730769230769 23	23.883982402116 6	7.5	9.923076923076923	11.471436098145958	9.0	12.653846153846153	13.841798460293539
4.0	8.714285714285714	12.942730999069928	25.0	27.523809523809526	22.234250712850766	8.0	12.761904761904763	12.300019357321199	17.0	21.190476190476 19	19.307042879786245
1.5	4.409090909090909	6.835893301495952	19.5	24.636363636363637	20.726209435778593	6.0	9.636363636363637	8.866982385421583	10.0	13.681818181818182	11.167070484395772

8. Histogram Plotting

```
os.makedirs('histograms', exist_ok=True)
for col in columns:
    if df[col].dropna().empty: continue
    clean_name = re.sub(r'[\W]+', '_', col)
    plt.hist(df[col].dropna(), bins=20, edgecolor='black')
    plt.title(f'{col} - All')
    plt.savefig(f"histograms/{clean_name}_all.png")
    plt.close()
    for team in df['Squad'].unique():
        subset = df[df['Squad'] == team][col].dropna()
        if subset.empty: continue
        safe_team = re.sub(r'[^\w]', '_', team)[:50]
        plt.hist(subset, bins=20, edgecolor='black')
        plt.title(f'{col} - {team}')
        plt.savefig(f"histograms/{clean_name}_{safe_team}.png")
        plt.close()
```

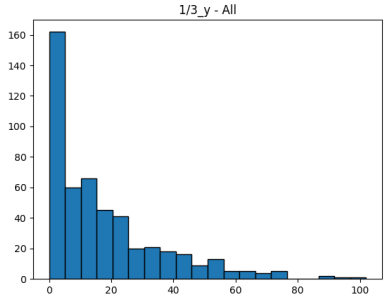
- **Purpose**: Creates histograms for each metric, both overall and per team, and saves them as PNG files in a histograms directory.
- **Logic**:

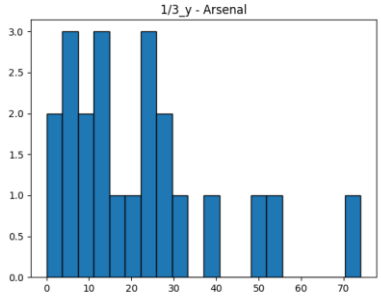
- Creates a directory named histograms if it doesn't exist (exist_ok=True prevents errors if the directory already exists).
- For each column in columns:
    - Skips the column if it has no valid data.
    - **Overall Histogram**:
        - Cleans the column name using re.sub to replace non-alphanumeric characters with underscores (e.g., 'xG' becomes 'xG').
        - Plots a histogram of the column's non-NaN values with 20 bins and black edges.
        - Sets the title as <column> - All (e.g., Gls - All).
        - Saves the plot as histograms/<clean_name>_all.png.

- Closes the plot to free memory.

- **Per-Team Histogram**:
  - Loops through each team in 'Squad'.
  - Filters the data for the current team and column, dropping NaN values.
  - Skips if the subset is empty.
  - Cleans the team name using re.sub to remove non-alphanumeric characters and limits it to 50 characters.
  - Plots a histogram for the subset with 20 bins and black edges.
  - Sets the title as <column> - <team> (e.g., Gls - Manchester United).
  - Saves the plot as histograms/<clean_name>_<safe_team>.png.
  - Closes the plot.

➢ **Outcome**: A set of histogram PNG files in the histograms directory, visualizing the distribution of each metric overall and per team, useful for identifying patterns or outliers. Some examples:





9. Best Performing Team Analysis

```
best_by_metric = {}
for col in columns:
    if df[col].dropna().empty: continue
    max_idx = df[col].idxmax()
    if pd.isna(max_idx): continue
    max_team = df.loc[max_idx, 'Squad']
    max_val = df[col].max()
    best_by_metric[col] = {'Team': max_team, 'Value': max_val}

count_by_team = pd.Series([v['Team'] for v in best_by_metric.values()]).value_counts()
best_team = count_by_team.idxmax()
num_top_metrics = count_by_team.max()

with open('best_team_analysis.txt', 'w', encoding='utf-8') as f:
    f.write("Teams with the highest value per metric:\n")
    for metric, info in best_by_metric.items():
        f.write(f"{metric}: {info['Team']} ({info['Value']})\n")
    f.write(f"\nSummary:\n{best_team} is the best team with {num_top_metrics} leading statistics.\n")
```

- **Purpose**: Identifies the team with the highest value for each metric and determines the overall best team based on the number of leading metrics.
- **Logic**:

- **Identify Best Team per Metric**:
    - Creates a dictionary best_by_metric to store the team with the highest value for each metric.
    - For each column:
        - Skips if the column has no valid data.
        - Finds the index of the maximum value (idxmax) and checks if it's valid (not NaN).
        - Retrieves the team (Squad) and maximum value for that index.
        - Stores the team and value in best_by_metric.
- **Count Leading Metrics per Team**:
    - Creates a pandas Series from the teams in best_by_metric and counts occurrences using value_counts.

- Identifies the team with the most leading metrics (idxmax) and the number of metrics it leads in (max).

- **Write Results**:
  - Opens best_team_analysis.txt in write mode with UTF-8 encoding.
  - Writes the team with the highest value for each metric (e.g., Gls: Manchester United (10)).
  - Writes a summary identifying the overall best team and the number of metrics it leads in (e.g., Manchester United is the best team with 3 leading statistics.).

➢ **Outcome**: A text file (best_team_analysis.txt) summarizing which team excels in each metric and identifying the overall best team based on the number of leading metrics.

best_team_analysis.txt

```
Teams with the highest value per metric:
Gls: Liverpool (28.0)
xG: Liverpool (24.0)
Sh: Brentford (51.0)
Tkl: Everton (124.0)
Int: West Ham (60.0)
Blocks: Brentford (71.0)

Summary:
Liverpool is the best team with 2 leading statistics.
```

PART III:

Use the K-means algorithm to classify players into groups based on their statistics.

• How many groups should the players be classified into? Why? Provide your comments

on the results.

• Use PCA to reduce the data dimensions to 2, then plot a 2D cluster of the data points.

I. Idea for the assignment:

· **Data Preparation**:

- Load results.csv and clean by replacing "N/a" with np.nan.
- Drop non-statistical columns (e.g., "Player", "Team").
- Select key statistics (e.g., "Gls", "Ast", "Tkl") for clustering.
- Handle missing values by filling with 0 (or another imputation method).
- Standardize the data using StandardScaler to ensure equal weighting of features.

· **Determine Optimal Number of Clusters (k)**:

- Use the Elbow method: Compute inertia for k from 1 to 10 and plot to identify the "elbow" point where inertia reduction slows.
- Use Silhouette Score: Calculate scores for k from 2 to 10 to measure cluster cohesion and separation.
- Choose k based on the plots (e.g., where the elbow bends or silhouette score peaks).

· **K-Means Clustering**:

- Apply K-Means with the chosen k to cluster players.
- Add cluster labels to the DataFrame for analysis.

· **PCA and Visualization**:

- Use PCA to reduce data to 2 dimensions for visualization.
- Plot a 2D scatter plot of the clusters, coloring by cluster and styling by simplified player position (e.g., "FW", "MF")

I. Idea for the assignment:

- **Data Preparation**: Load `results.csv`, clean missing values, select numeric features, and standardize them.
- **Clustering**:
  - Determine the optimal number of clusters (k) using Elbow & Silhouette methods.
  - Apply K-means clustering with optimal `k`.
- **PCA + Visualization**:
  - Use PCA to reduce data to 2 dimensions.
  - Visualize the clusters in a scatter plot.
- **Output**: Save plots, clustered data, and summary statistics.

II. Code Analysis

1. Library Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import os
from kneed import KneeLocator
```

- **Purpose: Load libraries needed for clustering, visualization, scaling, and file I/O.**
- **Key Tools:**
  - **StandardScaler: Standardize feature scales.**
  - **KMeans: Clustering algorithm.**
  - **silhouette_score: Measures quality of clusters.**
  - **PCA: Reduces dimensions for visualization.**
  - **KneeLocator: Identifies the "elbow" in the curve.**

2. Output Directory Setup

```
def setup_output_folder(base_path='D:/btl/results'):
    if not os.path.exists(base_path):
        os.makedirs(base_path)
    return base_path
```

- Creates the result folder if it doesn’t exist.
- Returns the path for storing outputs (images, CSVs).

3. Data Preprocessing

```
def preprocess_data(dataframe):
    numeric_columns = [col for col in dataframe.columns if col not in ['Name', 'Nation', 'Squad', 'Pos']]
    data = dataframe[numeric_columns].copy()
    data = data.apply(pd.to_numeric, errors='coerce').fillna(0)
    scaler = StandardScaler()
    return scaler.fit_transform(data)
```

- Removes non-numeric columns like Name, Nation, Squad, and Pos.
- Converts columns to numeric types and fills missing values with 0.
- Standardizes the data using StandardScaler for K-Means.

4. Metric Calculation for Optimal k

```python
def compute_metrics(scaled_data, k_values=range(2, 11)):
    distortions = []
    silhouette_values = []
    for k in k_values:
        model = KMeans(n_clusters=k, random_state=42, n_init=10)
        model.fit(scaled_data)
        distortions.append(model.inertia_)
        cluster_labels = model.labels_
        silhouette_values.append(silhouette_score(scaled_data, cluster_labels))
    return distortions, silhouette_values
```

For each k from 2 to 10:

- Fits a KMeans model.
- Stores:
    - Inertia (distortion): Used in the Elbow method.
    - Silhouette Score: Measures how well points fit within their clusters.

5. Visualization of Metrics

```python
def plot_metrics(k_values, distortions, silhouette_values, save_path):
    plt.figure(figsize=(12, 5))

    plt.subplot(121)
    plt.plot(k_values, distortions, 'bo-')
    plt.title('Elbow Curve')
    plt.xlabel('Clusters (k)')
    plt.ylabel('Distortion')
    plt.grid(True)

    plt.subplot(122)
    plt.plot(k_values, silhouette_values, 'go-')
    plt.title('Silhouette Metric')
    plt.xlabel('Clusters (k)')
    plt.ylabel('Score')
    plt.grid(True)

    plt.tight_layout()
    plt.savefig(os.path.join(save_path, 'metrics_plot.png'))
    plt.close()
```

➢ Plots two subplots:

- Left: Elbow curve (distortion vs. k).
- Right: Silhouette score vs. k.

➢ Saves the plot as metrics_plot.png.

6. K-Means Clustering

```
def apply_clustering(scaled_data, optimal_clusters):
    kmeans_model = KMeans(n_clusters=optimal_clusters, random_state=42, n_init=10)
    return kmeans_model.fit_predict(scaled_data)
```

- Applies KMeans using the best k.
- Returns cluster labels for each player.

7. PCA Visualization

```
def visualize_pca(scaled_data, cluster_labels, save_path):
    pca_model = PCA(n_components=2)
    reduced_data = pca_model.fit_transform(scaled_data)
    variance_ratio = pca_model.explained_variance_ratio_

    plt.figure(figsize=(10, 6))
    plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=cluster_labels, cmap='viridis', alpha=0.6)
    plt.title('PCA Visualization of Clusters')
    plt.xlabel(f'PC1 ({variance_ratio[0]*100:.1f}% variance)')
    plt.ylabel(f'PC2 ({variance_ratio[1]*100:.1f}% variance)')
    plt.grid(True)
    plt.savefig(os.path.join(save_path, 'pca_visualization.png'))
    plt.close()

    return variance_ratio
```

- Reduces high-dimensional data to 2D using PCA.
- Plots players as points, colored by cluster.
- Annotates axes with variance explained.
- Saves image as pca_visualization.png.
- Returns the variance ratio of PC1 and PC2

8.Cluster Summary Statistics

```
def summarize_clusters(dataframe, cluster_labels, save_path):
    dataframe['ClusterID'] = cluster_labels
    summary = dataframe.groupby('ClusterID').agg({
        'Pos': lambda x: x.mode()[0] if not x.mode().empty else 'Unknown',
        'Gls': 'mean',
        'Ast': 'mean',
        'xG': 'mean',
        'xAG': 'mean',
        'Min': 'mean',
        'Squad': lambda x: x.value_counts().index[0] if not x.value_counts().empty else 'Unknown'
    }).reset_index()
    summary.to_csv(os.path.join(save_path, 'cluster_stats.csv'), index=False)
    return summary
```

- **Adds a ClusterID column to the original dataset.**
- **Groups by ClusterID, calculates:**
  - Most common Pos and Squad.
  - Mean values of: Gls, Ast, xG, xAG, Min.
- **Saves result to cluster_stats.csv.**

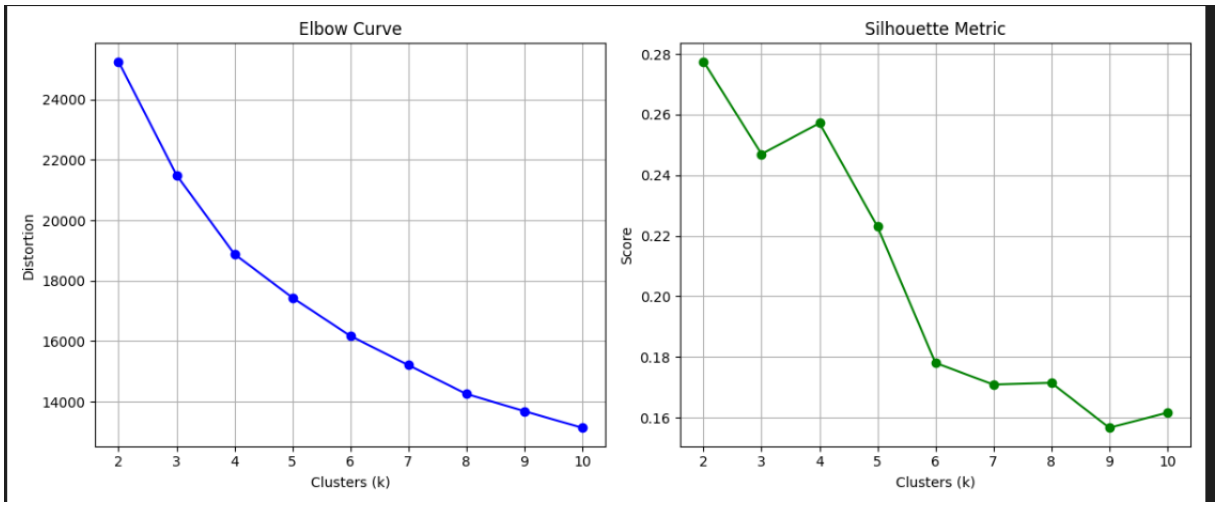
9.Main Function: Pipeline Controller

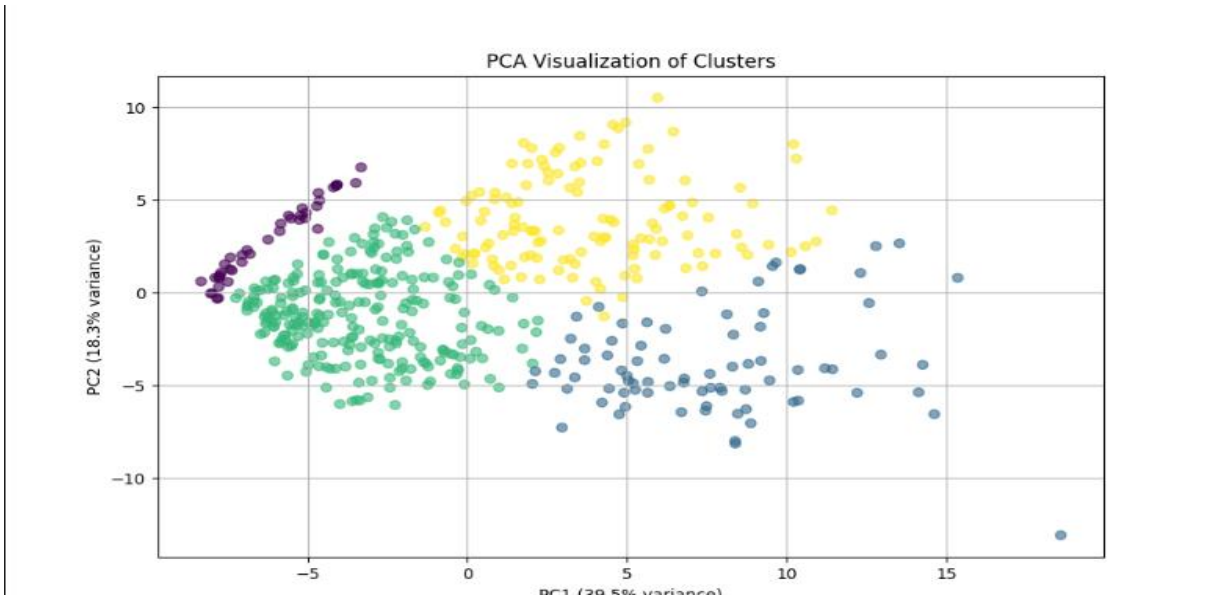
```
def run_analysis():
    output_dir = setup_output_folder()
    try:
        dataset = pd.read_csv('D:/btl/results.csv')
    except FileNotFoundError:
        print("Data file not found!")
        return
    if dataset.empty:
        print("Dataset is empty!")
        return
    processed_data = preprocess_data(dataset)
    k_range = range(2, 11)
    distortions, silhouette_values = compute_metrics(processed_data, k_range)
    plot_metrics(k_range, distortions, silhouette_values, output_dir)
    elbow_finder = KneeLocator(k_range, distortions, curve='convex', direction='decreasing')
    best_k = elbow_finder.knee
    cluster_assignments = apply_clustering(processed_data, best_k)
```

```
pca_variance = visualize_pca(processed_data, cluster_assignments, output_dir)
print(f"PCA Variance: {pca_variance.sum():.2f} (PC1: {pca_variance[0]:.2f}, PC2: {pca_variance[1]:.2f})")
cluster_stats = summarize_clusters(dataset, cluster_assignments, output_dir)
print(f"Clustered into {best_k} groups")
print("Cluster Stats:")
print(cluster_stats)
dataset.to_csv(os.path.join(output_dir, 'clustered_data.csv'), index=False)
print("Output saved in results folder")
```

Step-by-step:

- Create result folder.
- Load results.csv.
- Preprocess the dataset.
- Compute clustering metrics (distortion and silhouette).
- Plot evaluation metrics.
- Use KneeLocator to find the elbow point → best k.
- Apply KMeans clustering.
- Reduce data via PCA and visualize.
- Generate summary statistics per cluster.
- Save final dataset with ClusterID to clustered_data.csv.





10.Script Entry Point

```
if __name__ == "__main__":
    run_analysis()
```

Ensures the program executes only when run directly .

PART IV:

4.1 - Collect player transfer values for the 2024-2025 season from https://www.footballtransfers.com. Note that only collect for the players whose playing time is greater than 900 minutes

4.2 - Propose a method for estimating player values. How do you select feature and model?

I. Idea for the assignment:

1. **Define Objective and Data Sources**:
    - Objective: Collect football players' transfer values, combine with playing time data from a CSV, and handle missing values.
    - Sources: Main API for player lists, search API (e.g., footballtransfers.com), and CSV file (results.csv).
2. **Collect Data from Main API**:
    - Send POST requests to fetch player lists (paginated, e.g., 22 pages, 25 players/page).
    - Convert JSON to DataFrame, filter columns player_name, estimated_value.
    - Add delays between requests to avoid being blocked.
3. **Process CSV Data**:
    - Read results.csv, filter players with playing time > 900 minutes.
    - Standardize column names (e.g., rename Player to player_name).
4. **Combine Data**:
    - Merge DataFrames from API and CSV using pd.merge (on player_name, how="left").
    - Identify players with missing estimated_value (NaN).
5. **Handle Missing Values with Search API**:
    - Search for each missing player via the search API, using normalized names (unidecode).
    - Update estimated_value from JSON response, handle errors (HTTP, JSON, not found).
6. **Save Results**:
    - Save DataFrame to df_combined.csv with utf-8-sig encoding, excluding index.
7. **Optimize and Validate**:
    - Add delays, retries, and caching for optimization.

- Validate data (check integrity, detect anomalies).

II. Code Analysis:

1. Import Required Libraries

```
import requests
import pandas as pd
import time
import requests
import time
from urllib.parse import urlencode
import unidecode
```

- **requests**: Used to send HTTP POST requests to the website's API to fetch data.
- **pandas**: Handles tabular data manipulation, creating and working with DataFrames.
- **time**: Allows adding delays between requests (though not used directly in the code).
- **urlencode**: Encodes strings for URLs (not used in the code, possibly redundant).
- **unidecode**: Converts Unicode characters (e.g., accented letters) to ASCII equivalents, useful for handling player names with diacritics.

Analysis:

- The imports are appropriate for the task, but urlencode and time are unused, indicating potential cleanup.
- unidecode is critical for standardizing player names, ensuring compatibility with the API

2. Set Up URL and Headers for Main API Request

```
url = "https://www.footballtransfers.com/us/values/actions/most-valuable-football-players/overview"
headers = {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br, zstd",
    "Accept-Language": "vi,vi-VN;q=0.9,en-US;q=0.8,en;q=0.7",
    "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
    "Cookie": "FootballTransfers_Language=4; _cb=D7FNq7C1N35-C-0Vam; usprivacy=1N--; _gid=GA1.2.1717814782.1746813889; _cb_svref=external; _gat_gtag_UA_177878282_1=1; _c
    "Origin": "https://www.footballtransfers.com",
    "Referer": "https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league",
    "Sec-Ch-Ua": '"Chromium";v="136", "Google Chrome";v="136", "Not.A/Brand";v="99"',
    "Sec-Ch-Ua-Mobile": "?0",
    "Sec-Ch-Ua-Platform": '"Windows"',
    "Sec-Fetch-Dest": "empty",
    "Sec-Fetch-Mode": "cors",
    "Sec-Fetch-Site": "same-origin",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36",
    "X-Requested-With": "XMLHttpRequest"
}
```

- **URL**: The API endpoint for retrieving a list of the most valuable football players.
- **Headers**:
    - Mimics a Chrome browser on Windows to avoid server restrictions.
    - Includes fields like Cookie, User-Agent, and Referer to make the request appear legitimate.
    - Content-Type: application/x-www-form-urlencoded indicates the payload is URL-encoded form data.
    - X-Requested-With: XMLHttpRequest signals an AJAX request.

Analysis:

- The headers are well-crafted to bypass basic anti-scraping measures.
- The hardcoded Cookie may expire or be invalid on other systems, posing a risk.
- **Improvement**: Dynamically fetch cookies or test if cookies are necessary for the API.

3. Set Up Payload for Main API Request

```
payload = {
    "orderBy": "estimated_value",
    "orderByDescending": 1,
    "page": 1,
    "pages": 0,
    "pageItems": 25,
    "positionGroupId": "all",
    "mainPositionId": "all",
    "playerRoleId": "all",
    "age": "all",
    "countryId": "all",
    "tournamentId": 31
}
```

- **Payload**: Specifies parameters for filtering and sorting the player list.
    - orderBy: estimated_value and orderByDescending: 1: Sort by transfer value in descending order.
    - page: Current page number (updated in a loop).
    - pageItems: 25: Number of records per page (25 players).
    - tournamentId: 31: Likely refers to a specific league (e.g., Premier League, based on the Referer).
    - Other fields (positionGroupId, mainPositionId, etc.) set to "all" to retrieve all players without filtering by position, age, or country.

Analysis:

- The payload is well-structured for fetching a broad dataset.
- The hardcoded tournamentId limits the scope to one league, which may not be explicit in the code's documentation.

- Improvement: Add comments explaining tournamentId or make it configurable.

4. Collect Data from Main API

```
all_data = []
for i in range(1, 23):
    payload["page"] = i
    response = requests.post(url, headers=headers, data=payload)

    if response.status_code == 200:
        data = response.json()
        records = data["records"]
        df = pd.DataFrame(records)
        df_filtered = df[["player_name", "estimated_value"]]
        all_data.append(df_filtered)
    else:
        print("Error:", response.status_code)


final_df = pd.concat(all_data, ignore_index=True)
```

- **Purpose**: Fetch data from 22 pages (approximately 550 players, 25 per page).
- **Process**:
    - Loop through pages 1 to 22, updating page in the payload.
    - Send a POST request using requests.post().
    - If successful (status_code == 200):
        - Parse JSON response, extract records.
        - Convert to DataFrame, filter to player_name and estimated_value columns.
        - Append filtered DataFrame to all_data.
    - If failed, print the error code.

- Concatenate all DataFrames in all_data into final_df with ignore_index=True to reset indices.

Analysis:

- Efficiently collects and filters data, keeping only essential columns.
- Hardcoding 22 pages is risky if the number of pages changes.
- No delay between requests increases the risk of being rate-limited or blocked.
- **Improvements**:
  - Dynamically determine the number of pages from the API response (if available).
  - Add time.sleep(1) between requests to mimic human behavior.
  - Save raw JSON responses for debugging.

5. Merge with CSV Data

```
playtime = pd.read_csv('results.csv')
df_play_time_filtered = playtime[playtime["Min"] > 900]
df_play_time_filtered.rename(columns={"Player": "player_name"}, inplace=True)
df_combined = pd.merge(df_play_time_filtered, final_df, on="player_name",how="left")
df_no_value = df_combined[df_combined["estimated_value"].isna()]
```

- **Read CSV:**
  - results.csv contains playing time data (e.g., Min for minutes played, Player for player names).
- **Filter Data:**
  - Keep players with more than 900 minutes of playtime (playtime["Min"] > 900).
  - Rename Player column to player_name for consistency with final_df.
- **Merge Data:**

    - Use pd.merge to combine df_play_time_filtered and final_df on player_name with how="left":
        - Retain all players from df_play_time_filtered.
        - Add estimated_value from final_df, leaving NaN if no match.
- **Check Missing Values:**
    - df_no_value contains rows in df_combined where estimated_value is NaN.

Analysis:

- The merge is correctly implemented to prioritize CSV data while adding API data.
- Renaming columns ensures compatibility, but assumes Player and player_name formats match.
- **Improvements:**
    - Add fuzzy matching for player names to handle slight variations (e.g., "João" vs. "Joao").
    - Validate CSV data (e.g., check for missing or invalid Min values).

6. Handle Missing Values with Search API

```python
for index, row in df_combined[df_combined["estimated_value"].isna()].iterrows():
    player_name = row["player_name"]


    safe_player_name = unidecode.unidecode(player_name)

    url = "https://www.footballtransfers.com/us/search/actions/search"

    headers = {
        "Accept": "*/*",
        "Accept-Encoding": "gzip, deflate, br, zstd",
        "Accept-Language": "vi,vi-VN;q=0.9,en-US;q=0.8,en;q=0.7",
        "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
        "Origin": "https://www.footballtransfers.com",
        "Referer": f"https://www.footballtransfers.com/us/search?search_value={safe_player_name}",
        "Sec-Fetch-Dest": "empty",
        "Sec-Fetch-Mode": "cors",
        "Sec-Fetch-Site": "same-origin",
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36",
        "X-Requested-With": "XMLHttpRequest"
    }

    payload = {
        "search_page": 1,
        "search_value": safe_player_name,
        "players": 1,
        "teams": 1
    }
```

- **Purpose**: Fetch transfer values for players with missing estimated_value using the search API.
- **Preparation**:
    - Iterate over rows in df_combined where estimated_value is NaN.
    - Normalize player names with unidecode to remove diacritics (e.g., "João" to "Joao").
    - Set up a new URL for the search API: https://www.footballtransfers.com/us/search/actions/search.
    - Use similar headers as the main API, but dynamically update Referer with the normalized player name.
    - Payload includes:
        - search_value: Normalized player name.
        - players: 1 and teams: 1: Search for both players and teams, but take the first result.

Analysis:

- unidecode ensures robust handling of non-ASCII names.
- Dynamic Referer enhances request authenticity.
- **Improvement**: Add validation for safe_player_name to ensure it's not empty or malformed

7.Process Search API Response

```
try:
    response = requests.post(url, headers=headers, data=payload)

    if response.status_code == 200:
        json_data = response.json()

        if json_data.get("hits"):
            first_hit = json_data["hits"][0]["document"]
            player_value = first_hit.get("transfer_value")

            if player_value:
                df_combined.at[index, "estimated_value"] = player_value
                print(f"✅ Đã cập nhật {player_name} với giá {player_value}")
            else:
                print(f"⚠️ Không có giá cho {player_name}")
        else:
            print(f"❌ Không tìm thấy {player_name}")
    else:
        print(f"🔁 Lỗi khi gọi API cho {player_name}: {response.status_code}")
except Exception as e:
    print(f"🚨 Lỗi với {player_name}: {e}")
```

- **Process:**
    - Send a POST request to the search API.
    - If successful (status_code == 200):
        - Parse JSON response.
        - If results exist (json_data.get("hits")):
            - Extract the first result (hits[0]["document"]).
            - Get transfer_value (note: different key from estimated_value in main API).

            - If transfer_value exists, update estimated_value in df_combined at the given index.
            - Print success (✅) or no-value (⚠️) messages.
        - If no results, print not-found (❌) message.
    - If the request fails, print the error code (🔁).
    - Catch exceptions (e.g., network issues, invalid JSON) and print error (🚨).
- **Analysis:**
    - Error handling is robust, covering HTTP errors and exceptions.
    - Logging with emojis improves readability for tracking progress.
    - No delay between requests risks rate-limiting.
    - The difference in key names (estimated_value vs. transfer_value) could cause confusion.
- **Improvements:**
    - Add time.sleep(1) to avoid overwhelming the server.
    - Implement retries for failed requests using a library like tenacity.
    - Standardize key names or document the difference clearly.
    - Validate transfer_value (e.g., ensure it's a positive number).

8. Save Results

```
df_combined.to_csv("df_combined.csv", index=False, encoding="utf-8-sig")
```

- Save df_combined (with updated transfer values) to df_combined.csv.
- index=False: Exclude the index column.

- encoding="utf-8-sig": Use UTF-8 with BOM for compatibility with tools like Excel, supporting special characters.

Analysis:

- The output format is suitable for further analysis or sharing.
- Improvement: Add a backup step to save intermediate results before overwriting.

4.2 - Propose a method for estimating player values. How do you select feature and model?

I. Idea for the assignment:

- Loads player data from df_combined.csv, which combines statistical data from results.csv (from Part I) and transfer values from part4.1.py.
- Preprocesses the data by handling missing features, parsing Age and estimated_value, and computing per-90 metrics.
- Trains a **Random Forest Regressor** to estimate transfer values based on selected features.
- Evaluates the model using Mean Absolute Error (MAE), R^2 Score, and Root Mean Squared Error (RMSE).
- Saves predictions, feature importance, and a visualization of the top features.

II. Code Analysis:

1. Imports

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
import matplotlib.pyplot as plt
import re
```

- **Purpose**:
    - pandas and numpy: Handle data manipulation and numerical operations.
    - sklearn modules: Support model training (RandomForestRegressor), preprocessing (StandardScaler, OneHotEncoder, ColumnTransformer), splitting data (train_test_split), cross-validation (cross_val_score), and evaluation (mean_absolute_error, r2_score, mean_squared_error).
    - matplotlib.pyplot: Create a feature importance plot.
    - re: Parse strings (e.g., estimated_value like '€32.2M').
- **Strengths**:
    - Comprehensive imports cover all necessary functionality for data processing, modeling, and visualization.
- **Weaknesses**:
    - No unused imports, but compatibility issues with older scikit-learn versions were addressed (e.g., manual RMSE calculation).

2. Helper Functions

parse_age

```
def parse_age(age_str):
    try:
        if isinstance(age_str, str) and "-" in age_str:
            years, days = map(int, age_str.split("-"))
            return years + (days / 365.0)  # Convert to decimal years
        return float(age_str)
    except (ValueError, TypeError):
        return np.nan
```

- **Purpose**: Converts Age from 'years-days' format (e.g., '27-165') to a float (e.g., 27.45 years).

- **Process:**
    - Checks if input is a string with a hyphen.
    - Splits into years and days, converts to decimal years.
    - Attempts to convert non-hyphenated strings to float.
    - Returns np.nan for invalid inputs.
- **Strengths:**
    - Robustly handles the specific format from https://foref.com.
    - Gracefully manages errors with try-except**.**
- **Weaknesses:**
    - Assumes days are divided by 365 (not accounting for leap years), but this is minor for transfer value estimation.

parse_transfer_value

```
def parse_transfer_value(value):
    try:
        if isinstance(value, str):
            value = re.sub(r'[€£M]', '', value).strip()
            return float(value)
        return float(value)
    except (ValueError, TypeError):
        return np.nan
```

- **Purpose**: Converts estimated_value from strings like '€32.2M' to floats (e.g., 32.2).
- **Process**:
    - Removes currency symbols (€, £) and M (for millions) using regex.
    - Converts the cleaned string to float.

- Handles non-string inputs by attempting float conversion.
- Returns np.nan for invalid inputs.

- **Strengths**:
  - Handles multiple currency formats (€, £) and the M suffix.
  - Robust error handling prevents crashes.
- **Weaknesses:**
  - May fail for unexpected formats (e.g., '32.2K' or '€32,2M' with commas). Could be extended with more regex patterns if needed.

3. Data Loading and Initial Preprocessing

```
df = pd.read_csv("df_combined.csv")
print("Available columns in df_combined.csv:", df.columns.tolist())
df["estimated_value"] = df["estimated_value"].apply(parse_transfer_value)
df = df.dropna(subset=["estimated_value"])
df["Age"] = df["Age"].apply(parse_age)
df["Age"] = df["Age"].fillna(df["Age"].median())
```

- **Purpose**: Loads df_combined.csv and preprocesses estimated_value and Age.
- **Process**:
  - Loads the CSV and logs columns for debugging.
  - Parses estimated_value to numerical values.
  - Drops rows with missing estimated_value (ensures valid target).
  - Parses Age and imputes missing values with the median.
- **Strengths**:
  - Debugging print statement helps identify available features.
  - Ensures the target (estimated_value) is numerical and complete.

- Median imputation for Age is reasonable for numerical features.

- **Weaknesses**:
  - Dropping rows with missing estimated_value may reduce the dataset size. Could consider imputation for missing values if needed.

4. Feature Selection

```
potential_numerical_features = [
    "Age", "Min", "MP", "Starts", "Gls", "Ast", "xG", "xAG",
    "YCrd", "RCrd", "PrgC", "PrgP", "PrgR", "Gls/90", "Ast/90", "xG/90", "xAG/90",
    "Cmp", "Tkl", "Int", "Blocks", "Touches", "Att 3rd", "Att Pen", "Succ%",
    "PrgDist", "CPA", "Recov", "Won%"
]
potential_categorical_features = ["Position", "Pos"]

available_columns = df.columns.tolist()
numerical_features = [f for f in potential_numerical_features if f in available_columns]
categorical_features = [f for f in potential_categorical_features if f in available_columns]

missing_features = [f for f in potential_numerical_features + potential_categorical_features if f not in available_columns]
if missing_features:
    print(f"Warning: Missing features in df_combined.csv: {missing_features}")

if "Gls/90" not in available_columns and "Gls" in available_columns and "Min" in available_columns:
    df["Gls/90"] = df["Gls"] / (df["Min"] / 90.0)
    numerical_features.append("Gls/90")
if "Ast/90" not in available_columns and "Ast" in available_columns and "Min" in available_columns:
    df["Ast/90"] = df["Ast"] / (df["Min"] / 90.0)
    numerical_features.append("Ast/90")

if not numerical_features and not categorical_features:
    raise ValueError("No valid features available in df_combined.csv. Please check the input data.")
```

- **Purpose: Selects features for the model, handling missing columns dynamically.**
- **Process:**
  - Defines a comprehensive list of potential numerical and categorical features based on the assignment's Part I requirements.
  - Filters features to those present in df_combined.csv.
  - Prints a warning for missing features.
  - Computes Gls/90 and Ast/90 if missing, using Gls, Ast, and Min.
  - Raises an error if no valid features are available.
- **Strengths:**

- Dynamic feature selection ensures the code runs even with incomplete data.
- Computing per-90 metrics enhances the feature set when base metrics are available.
- Warning message aids debugging by identifying missing features.

- **Weaknesses:**
  - Many expected features (e.g., YCrd, PrgC, Position) are missing, suggesting issues with results.csv from Part I.
  - Could compute additional per-90 metrics (e.g., xG/90) if xG is available.
  - Assumes column names match exactly; fuzzy matching could handle variations (e.g., Yellow Cards vs. YCrd).

5. Feature Preprocessing

```
for col in numerical_features:
    df[col] = pd.to_numeric(df[col], errors="coerce")
    df[col] = df[col].replace("N/a", 0).fillna(df[col].median())
```

- **Purpose: Ensures numerical features are properly formatted.**
- **Process:**
  - Converts columns to numeric, coercing invalid values to NaN.
  - Replaces N/a with 0 (suitable for performance metrics like Gls).
  - Imputes NaN with the median for robustness.
- **Strengths:**
  - Robustly handles non-numerical values and missing data.
  - Median imputation is appropriate for skewed numerical features**.**

- **Weaknesses:**
    - N/a with 0 may not be ideal for Replacing all features (e.g., Succ%); context-specific imputation could be better.
    - Does not validate data ranges (e.g., negative Gls).

6. Model Setup

```
X = df[numerical_features + categorical_features]
y = df["estimated_value"]
transformers = [("num", StandardScaler(), numerical_features)]
if categorical_features:
    transformers.append(("cat", OneHotEncoder(drop="first", handle_unknown="ignore"), categorical_features))

preprocessor = ColumnTransformer(transformers=transformers)

model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("regressor", RandomForestRegressor(n_estimators=100, max_depth=10, min_samples_split=5, random_state=42))
])
```

➢ **Purpose**: Prepares features (X) and target (y), and sets up a modeling pipeline.

➢ **Process**:

- Selects features (X) and target (y).
- Creates a ColumnTransformer to:
    - Scale numerical features with StandardScaler.
    - One-hot encode categorical features (if present) with OneHotEncoder, dropping the first category to avoid multicollinearity.
- Builds a Pipeline combining preprocessing and a RandomForestRegressor with 100 trees, max depth of 10, and minimum split of 5.

➢ **Strengths**:

- Pipeline ensures consistent preprocessing for training and prediction.
- Random Forest is well-suited for non-linear relationships and robust to outliers.
- Categorical preprocessing is optional, handling cases where Position is missing.

➢ **Weaknesses**:

- Hyperparameters (n_estimators=100, max_depth=10) are not tuned; grid search could optimize performance.
- Limited features (due to missing columns) may reduce model accuracy.

7. Model Training and Evaluation

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print(f"Model Performance on Test Set:")
print(f"Mean Absolute Error (MAE): {mae:.2f} million euros")
print(f"R² Score: {r2:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f} million euros")
```

➢ **Purpose**: Trains the model and evaluates performance.

➢ **Process**:

- Splits data into 80% training and 20% testing sets.
- Fits the model on the training set.
- Predicts on the test set and computes:
    - **MAE**: Average absolute error in millions of euros.
    - **R^2**: Proportion of variance explained.
    - **RMSE**: Square root of mean squared error, computed manually for compatibility.

➢ **Strengths**:

- Uses standard regression metrics (MAE, R^2, RMSE) to assess performance.

- Manual RMSE calculation ensures compatibility with older scikit-learn versions.

➢ **Weaknesses**:

- No validation of metric values (e.g., negative R^2 indicates poor fit).
- Limited features may lead to lower R^2 or higher errors.

8. Cross-Validation

```
cv_scores = cross_val_score(model, X, y, cv=5, scoring="r2")
print(f"5-Fold Cross-Validation R² Scores: {cv_scores}")
print(f"Average CV R² Score: {cv_scores.mean():.2f} ± {cv_scores.std():.2f}")
```

➢ **Purpose**: Assesses model robustness using 5-fold cross-validation.

➢ **Process**:

- Computes R^2 scores for 5 folds.
- Prints individual scores and the mean ± standard deviation.

➢ **Strengths**:

- Cross-validation provides a robust estimate of model performance.
- Reporting mean and standard deviation highlights consistency.

➢ **Weaknesses**:

- Only uses R^2; could include MAE or RMSE for a fuller picture.
- Small dataset size may lead to high variance in CV scores

9. Predictions and Saving Results

```
df["predicted_value"] = model.predict(X)
df[["player_name", "estimated_value", "predicted_value"]].to_csv(
    "estimated_values.csv", index=False, encoding="utf-8-sig"
)
```

- **Purpose**: Generates predictions for all players and saves them.
- **Process**:

- Predicts transfer values for the entire dataset.
- Saves player_name, estimated_value, and predicted_value to estimated_values.csv with UTF-8-SIG encoding (Excel-compatible).

- **Strengths**:

- Saves essential columns for analysis.
- UTF-8-SIG encoding supports special characters in player names.

- **Weaknesses**:

- Does not save additional stats, which could be useful for debugging or reporting.

10. Feature Importance

```
feature_names = numerical_features
if categorical_features:
    feature_names += list(model.named_steps["preprocessor"].transformers_[1][1].get_feature_names_out())
importances = model.named_steps["regressor"].feature_importances_
feature_importance_df = pd.DataFrame({
    "Feature": feature_names,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)
feature_importance_df.to_csv("feature_importance.csv", index=False, encoding="utf-8-sig")
```

- **Purpose**: Extracts and saves feature importance scores.
- **Process**:

- Combines numerical and one-hot-encoded categorical feature names.

- Retrieves importance scores from the Random Forest model.
- Creates a DataFrame, sorts by importance, and saves to feature_importance.csv.

➢ **Strengths**:

- Provides insights into which features drive transfer value predictions.
- Saves results for inclusion in the report.

➢ **Weaknesses**:

- Importance scores may be less meaningful with few features.

11. Visualization

```
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df["Feature"][:10], feature_importance_df["Importance"][:10])
plt.xlabel("Feature Importance")
plt.title("Top 10 Features Influencing Player Transfer Value")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.savefig("feature_importance_plot.png")
plt.close()
```

- **Purpose**: Creates a bar plot of the top 10 features by importance.
- **Process**:
  - Plots a horizontal bar chart with feature names and importance scores.
  - Inverts the y-axis for readability and saves as feature_importance_plot.png.
- **Strengths**:
  - Visualizes key predictors, enhancing the report.
  - Clear formatting with appropriate size and labels.
- **Weaknesses**:

- Limited to top 10 features; could be configurable.
- No additional plots (e.g., actual vs. predicted values).