POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

DEPARTMENT OF PYTHON PROGRAMMING LANGUAGE

FINAL ASSIGNMENT
FOOTBALL PLAYER DATA COLLECTION

INSTRUCTOR	: KIM NGUYEN BACH
STUDENT NAME	: TRAN TRUNG KIEN
STUDENT ID	: B23DCCE058
CLASS	: D23CQCE04-B
GROUP	: 04

Hanoi – 2025

Table of Contents

1. Overview and Objectives

This assignment requires students to use the Python programming language to collect, process, and analyze statistical data on football players participating in the English Premier League during the 2024–2025 season. It offers a valuable hands-on opportunity for students to engage in a real-world data analysis process—particularly within the sports domain, where data plays a vital role in performance assessment and strategic decision-making.

Throughout the assignment, students will learn how to collect data from a specialized football statistics website (fbref.com), handle raw and inconsistent data, and organize and store it in a structured format. Data manipulation skills will be developed through cleaning, standardizing, and merging various data sources—especially when dealing with complex statistics comprising dozens of metrics per player.

Once the data has been prepared, students will perform basic statistical analysis such as calculating averages, medians, standard deviations, and identifying standout or underperforming players based on each metric. Furthermore, visualizing the data through histograms will enable a deeper understanding of the distribution of various statistics across the league and within each team.

At a more advanced level, students will apply machine learning algorithms such as K-means clustering to group players according to performance characteristics. Dimensionality reduction using PCA (Principal Component Analysis) is also utilized to present the data in a clear and interpretable two-dimensional plot.

An additional component of the project involves collecting and analyzing the transfer market values of players from other sources(footballtransfers.com). Based on this information, students are expected to propose a method to estimate player market values using available statistical indicators. This requires a sound understanding of feature selection and model design within the context of real-world data.

The ultimate goal of this assignment is not only to develop a functional program but also to foster analytical thinking, problem-solving skills, and the ability to clearly and meaningfully present data insights. These are essential skills for anyone pursuing a career in data science, sports analytics, or practical software development.

2. Data Collection from FBref

2.1. Objective

To collect detailed statistical data of players participating in the 2024–2025 Premier League season from the website FBref.com. Only data from players who have played more than 90 minutes is considered.

2.2. Tools Used

Programming Language: Python

Libraries:

- **Selenium**: Automates Chrome browser to access and extract the HTML content of statistical tables.
- **BeautifulSoup**: Parses and processes HTML.
- **Pandas**: Handles and standardizes data.
- **Logging**: Records the program's execution process.

- **WebDriver Manager**: Automatically installs the appropriate ChromeDriver.

```
import os
import time
import logging
import pandas as pd
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, NoSuchElementException
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import random
from selenium.webdriver.chrome.options import Options
```

2.3. Procedure

- **Identify data sources**: Eight statistical tables are used: standard, keeper, shooting, passing, gca, defense, possession, and misc. Each table has its own URL and ID.

- **Access and collect data**:

  - Use a headless Chrome browser to automatically access each statistical table.

  - Ensure the page is fully loaded and all data tables are displayed completely.

- Extract the HTML of the tables, then parse the thead and tbody sections to obtain headers and row data.

```python
table_html = table.get_attribute('outerHTML')
soup = BeautifulSoup(table_html, 'html.parser')

# Get all header rows
headers = []
thead = soup.find('thead')
if thead:
    # Get column groups (top header row)
    colgroups = []
    for tr in thead.find_all('tr'):
        if 'over_header' in tr.get('class', []):
            for th in tr.find_all(['th', 'td']):
                colspan = int(th.get('colspan', 1))
                colgroups.extend([th.text.strip()] * colspan)

    # Get stat columns (bottom header row)
    stat_headers = []
    for tr in thead.find_all('tr'):
        if 'over_header' not in tr.get('class', []):
            for th in tr.find_all(['th', 'td']):
                if 'data-stat' in th.attrs:
                    stat_headers.append(th['data-stat'])
                else:
                    stat_headers.append(th.text.strip())

    # Combine headers
    if len(colgroups) == len(stat_headers):
        headers = [f"{cg}_{sh}" if cg else sh for cg, sh in zip(colgroups, stat_headers)]
    else:
        headers = stat_headers

if not headers:
    raise ValueError("No headers found in table")

# Log available headers for debugging
logging.info(f"Available headers: {headers}")

# Get data rows
rows = []
tbody = soup.find('tbody')
if not tbody:
    raise ValueError("No tbody found in table")

for tr in tbody.find_all('tr'):
    # Skip header rows in tbody
    if 'class' in tr.attrs and ('thead' in tr['class'] or 'spacer' in tr['class']):
        continue

    row = []
    tds = tr.find_all(['td', 'th'])
```

- Apply special processing for columns such as player, age, and nationality to standardize formats.

```
# Process each cell
for td in tds:
    if 'data-stat' in td.attrs:
        # Get the text content
        text = td.text.strip()
        # Handle special cases
        if td['data-stat'] == 'player':
            # Get player name and remove any special characters
            text = ' '.join(text.split())
        elif td['data-stat'] == 'nationality':
            # Keep exact nationality code (e.g., 'ENG', 'FRA')
            parts = td.text.strip().split()
            if parts:
                text = parts[-1]  # Get the last part (country code)
            else:
                text = ""
        elif td['data-stat'] == 'age':
            # Get age from data-age attribute if available
            text = td.get('data-age', td.text.strip())
            # If data-age is not available, try to get from title attribute
            if not text:
                text = td.get('title', td.text.strip())
            # If neither is available, use the text content
            if not text:
                text = td.text.strip()
        elif td['data-stat'] in ['team', 'position']:
            # Keep original formatting for these fields
            text = td.text.strip()
        else:
            # For numeric fields, keep only digits, decimal points, and minus signs
            text = ''.join(c for c in text if c.isdigit() or c in '.-')
            text = text if text else "N/a"

        row.append(text)
    else:
        row.append(td.text.strip())
```

- Retain only players who have played more than 90 minutes.

```
def merge_stats(self, stats_dict):
    """Merge different types of statistics into a single DataFrame"""
    try:
        if 'standard' not in stats_dict or stats_dict['standard'] is None:
            logging.error("No standard stats available to merge")
            return None

        result_df = stats_dict['standard'].copy()
        logging.info(f"Starting merge with standard stats. Shape: {result_df.shape}")

        # Basic column renaming
        result_df = result_df.rename(columns=self.column_mappings)

        # Filter minutes
        if 'Minutes' in result_df.columns:
            result_df['Minutes'] = pd.to_numeric(
                result_df['Minutes'].astype(str).str.replace(r'[^\d.-]', '', regex=True),
                errors='coerce'
            ).fillna(0)
            result_df = result_df[result_df['Minutes'] > 90]
```

- **Data standardization**:

- Use a column_mappings dictionary to rename columns to a unified format (e.g., "Performance_goals" → "Goals").

```python
# Define required statistics
self.required_stats = {
    'standard': [
        'player',
        'nationality',  # Nation
        'team',         # Team
        'position',     # Position
        'age',          # Age
        'Playing Time_games',        # matches played
        'Playing Time_games_starts', # starts
        'Playing Time_minutes',      # minutes
        'Performance_goals',         # goals
        'Performance_assists',       # assists
        'Performance_cards_yellow',  # yellow cards
        'Performance_cards_red',     # red cards
        'Expected_xg',              # expected goals (xG)
        'Expected_xg_assist',       # expected Assist Goals (xAG)
        'Progression_progressive_carries',  # PrgC
        'Progression_progressive_passes',   # PrgP
        'Progression_progressive_passes_received',  # PrgR
        'Per 90 Minutes_goals_per90',      # Gls per 90
        'Per 90 Minutes_assists_per90',    # Ast per 90
        'Per 90 Minutes_xg_per90',         # xG per 90
        'Per 90 Minutes_xg_assist_per90'   # xGA per 90
    ],
    'keeper': [
        'Performance_gk_goals_against_per90',  # GA90
        'Performance_gk_save_pct',             # Save%
        'Performance_gk_clean_sheets_pct',     # CS%
        'Penalty Kicks_gk_pens_save_pct'       # Penalty kicks Save%
    ],
```

- Remove duplicate entries and fill in missing values with "N/a".

```python
# Merge các loại thống kê khác
    for stat_type in ['keeper', 'shooting', 'passing', 'gca', 'defense', 'possession', 'misc']:
        if stat_type in self.stats_data:
            try:
            # Gọi process_player_stats cho từng loại thống kê
                df = self.process_player_stats(
                    self.stats_data[stat_type],
                    self.required_stats[stat_type]
                )
                if df is not None:
                # Merge vào base_df
                    base_df = pd.merge(
                        base_df,
                        df,
                        on=['Player', 'Team'],
                        how='left'
                    )
            except Exception as e:
                logging.error(f"Error merging {stat_type}: {str(e)}")
                continue

    # Fill missing values with "N/a"
    base_df = base_df.fillna("N/a")
```

- **Data merging**:
    - Merge data from other tables with the standard table based on Player and Team.

2.4. Results Obtained

Output file: results.csv

Data overview:

- Over 78 statistical columns covering all categories: basic information, technical metrics, attacking, defensive, ball control, duels, etc.
- Each row represents a player with name, team, nationality, position, age, number of matches, playing time, etc.
- Key performance indicators such as xG, Ast/90, SoT%, Tkl, Int, Recov, Won%, etc., are included.

Player	Nation	Team	Position	Age	MP	Starts	Minutes	Goals	Assists	Yellow Car	Red Cards	xG	xAG	PrgC	PrgP	PrgR	Gls/90	Ast/90	xG/90	xGA/90	GA90	Save%	CS%	PKsv%	SoT%	SoT/90	G/Sh	Dist	Cmp	Cmp%
Aaron Cre	ENG	West Ham	DF	35-143	15	8	676	0	0	2	0	0.1	1.1	4	28	2	0	0	0.01	0.15	N/a	N/a	N/a	N/a	0	0	0	29.8	415	81.5
Aaron Ran	ENG	Southampt	GK	26-358	27	27	2430	0	0	2	0	0	0	0	0	0	0	0	0	0	2.3	67.1	7.4	28.6	N/a	0	N/a	N/a	715	64.6
Aaron Wa	ENG	West Ham	DF	27-162	33	32	2884	2	3	1	0	1.2	3.2	101	132	150	0.06	0.09	0.04	0.1	N/a	N/a	N/a	N/a	31.3	0.16	0.13	15.1	1321	80.5
Abdoulaye	MLI	Everton	MF	32-126	30	29	2425	3	1	5	1	3.9	2.3	40	78	91	0.11	0.04	0.14	0.09	N/a	N/a	N/a	N/a	31	0.33	0.1	13.8	676	79.6
Abdukodir	UZB	Manchest	DF	21-067	6	6	503	0	0	1	0	0	0.1	1	25	2	0	0	0	0.02	N/a	N/a	N/a	N/a	0	0	0	23.9	360	91.1
Abdul Fata	GHA	Leicester C	FW	21-060	11	6	579	0	2	0	0	0.4	1.6	42	17	60	0	0.31	0.06	0.24	N/a	N/a	N/a	N/a	38.5	0.78	0	31.2	123	62.4
Adam Arm	ENG	Southampt	FW,MF	28-086	20	15	1248	2	2	4	0	3.3	1.2	25	21	79	0.14	0.14	0.24	0.09	N/a	N/a	N/a	N/a	28	0.5	0.08	16.6	231	80.2
Adam Lalla	ENG	Southampt	MF	36-362	14	5	361	0	2	4	0	0.2	0.9	6	24	10	0	0.5	0.04	0.23	N/a	N/a	N/a	N/a	0	0	0	18.1	221	85
Adam Smit	ENG	Bournemo	DF	34-008	22	17	1409	0	0	6	0	0.7	0.3	12	40	31	0	0	0.04	0.02	N/a	N/a	N/a	N/a	0	0	0	16.9	436	75.7
Adam Wel	ENG	Brighton	DF	30-123	11	8	617	0	0	0	0	0	0.5	7	40	2	0	0	0	0.07	N/a	N/a	N/a	N/a	100	0.15	0	15.3	429	85.6
Adam Wha	ENG	Crystal Pal	MF	20-339	20	16	1318	0	2	2	0	0.4	3	14	107	11	0	0.14	0.03	0.21	N/a	N/a	N/a	N/a	38.5	0.34	0	25.4	548	75.6
Adama Tra	ESP	Fulham	FW,MF	29-102	33	16	1592	2	6	3	0	3.9	4.7	89	61	145	0.11	0.34	0.22	0.27	N/a	N/a	N/a	N/a	29.7	0.62	0.05	14.3	386	72
Albert Grø	DEN	Southampt	FW,MF	23-349	4	2	143	0	0	0	0	0.1	0	1	1	3	0	0	0.07	0.01	N/a	N/a	N/a	N/a	0	0	0	22.4	13	72.2
Alejandro	ARG	Manchest	MF,FW	20-310	34	23	2146	6	2	3	0	7.2	4.5	139	56	281	0.25	0.08	0.3	0.19	N/a	N/a	N/a	N/a	34.9	1.22	0.07	17.2	635	79.1
Alex Iwobi	NGA	Fulham	FW,MF	29-004	35	33	2796	9	6	1	0	4.6	6.9	135	199	224	0.29	0.19	0.15	0.22	N/a	N/a	N/a	N/a	44.4	0.9	0.14	18.2	1057	74.4
Alex McCa	ENG	Southampt	GK	35-155	5	5	450	0	0	0	0	0	0	0	0	0	0	0	0	0	2.6	70.3	0	0	N/a	0	N/a	N/a	117	75
Alex Palme	ENG	Ipswich To	GK	28-270	11	11	990	0	0	2	0	0	0	0	1	0	0	0	0	0	2.45	58.7	0	0	N/a	0	N/a	N/a	241	67.7
Alex Scott	ENG	Bournemo	MF	21-259	18	7	709	0	0	2	0	0.7	0.8	15	49	33	0	0	0.08	0.11	N/a	N/a	N/a	N/a	38.5	0.63	0	20	265	83.1
Alexander	SWE	Newcastle	FW	25-228	32	32	2577	23	6	1	0	19.9	4.3	81	80	200	0.8	0.21	0.69	0.15	N/a	N/a	N/a	N/a	42	1.29	0.22	14.6	479	74.5
Alexis Mac	ARG	Liverpool	MF	26-134	34	30	2575	5	5	6	0	2.8	4.6	34	174	79	0.17	0.17	0.1	0.16	N/a	N/a	N/a	N/a	34.2	0.45	0.13	18.1	1260	83.5
Ali Al Ham	IRQ	Ipswich To	FW	23-067	11	0	134	0	0	3	0	0.4	0.1	4	3	14	0	0	0.24	0.05	N/a	N/a	N/a	N/a	0	0	0	15.7	25	83.3
Alisson	BRA	Liverpool	GK	32-217	25	25	2238	0	0	0	0	0	0.6	0	0	0	0	0	0	0.02	0.92	71.4	36	0	N/a	0	N/a	N/a	681	83.6
Alphonse A	FRA	West Ham	GK	32-069	24	23	2080	0	0	0	0	0	0	0	0	0	0	0	0	0	1.69	62.9	17.4	N/a	N/a	0	N/a	N/a	633	69.3
Altay Bayı	TUR	Manchest	GK	27-023	2	2	180	0	0	0	0	0	0	0	0	0	0	0	0	0	4	33.3	0	N/a	N/a	0	N/a	N/a	61	55.5
Amad Diall	CIV	Manchest	FW,MF	22-300	23	17	1639	7	6	3	0	4.2	4	93	55	163	0.38	0.33	0.23	0.22	N/a	N/a	N/a	N/a	32.6	0.77	0.16	18	676	82.9
Amadou O	BEL	Aston Villa	MF	23-264	23	17	1378	3	0	4	0	2.1	0.3	21	60	14	0.2	0	0.14	0.02	N/a	N/a	N/a	N/a	35.3	0.39	0.18	14.7	607	88.5
Andreas Pe	BRA	Fulham	MF	29-126	31	23	1879	2	4	7	0	3.5	4.5	27	99	78	0.1	0.19	0.17	0.21	N/a	N/a	N/a	N/a	22.9	0.38	0.06	22.1	641	68.9
Andrew Ro	SCO	Liverpool	DF	31-057	31	27	2308	0	0	3	1	1	4.1	58	165	95	0	0	0.04	0.16	N/a	N/a	N/a	N/a	15.4	0.08	0	14.5	1456	83
André	BRA	Wolves	MF	23-295	30	28	2249	0	0	7	0	0.4	0.7	8	95	11	0	0	0.02	0.03	N/a	N/a	N/a	N/a	11.1	0.04	0	25.3	1111	92.8
André Ona	CMR	Manchest	GK	29-035	33	33	2970	0	0	0	0	0	0.2	0	1	0	0	0	0	0.01	1.3	69	27.3	25	N/a	0	N/a	N/a	920	73.1
Andrés Ga	ESP	Aston Villa	DF,MF	22-089	7	5	318	0	0	0	0	0	0.2	26	15	28	0	0	0.01	0.05	N/a	N/a	N/a	N/a	0	0	0	26.6	174	86.6
Andy Irving	SCO	West Ham	MF	24-359	10	1	166	0	0	2	0	0.1	0.2	0	9	4	0	0	0.06	0.13	N/a	N/a	N/a	N/a	50	1.08	0	26.2	109	87.9
Anthony El	SWE	Nott'ham	FW,MF	23-010	35	28	2232	6	9	1	0	4.2	4.8	87	54	190	0.24	0.36	0.17	0.19	N/a	N/a	N/a	N/a	52.5	0.85	0.15	17.3	479	65.2
Anthony G	ENG	Newcastle	FW	24-072	31	25	2235	6	5	2	0	8	4.9	111	93	197	0.24	0.2	0.32	0.2	N/a	N/a	N/a	N/a	28.6	0.64	0.09	16.8	671	73.3
Antoine Se	GHA	Bournemo	FW	25-120	34	33	2933	9	5	9	0	9.3	5.8	133	106	259	0.28	0.15	0.29	0.18	N/a	N/a	N/a	N/a	31.4	1.14	0.08	17.2	737	73.2
Antonee R	USA	Fulham	DF	27-272	34	33	2989	0	10	8	0	0.7	4.1	114	123	265	0	0.3	0.02	0.12	N/a	N/a	N/a	N/a	12.5	0.06	0	22.1	1523	76.6
Antony	BRA	Manchest	DF,MF	25-072	8	0	141	0	0	0	0	1	0.1	5	6	7	0	0	0.65	0.08	N/a	N/a	N/a	N/a	50	1.91	0	15.8	56	81.2
Antonín Ki	CZE	Tottenham	GK	22-055	4	4	360	0	0	0	0	0	0.1	1	1	0	0	0	0	0.02	1.75	56.3	25	N/a	N/a	0	N/a	N/a	116	80
Archie Gra	ENG	Tottenham	DF,MF	19-056	25	16	1481	0	0	0	0	0	0.1	6	34	15	0	0	0	0.01	N/a	N/a	N/a	N/a	100	0.06	0	24.6	939	88.2
Arijanet M	KVX	Ipswich To	GK	26-181	18	18	1620	0	0	1	0	0	0	0	1	0	0	0	0	0	1.83	70	5.6	0	N/a	0	N/a	N/a	454	71.7
Armando E	ALB	Everton	FW	23-239	10	4	334	0	0	1	0	0.1	0.2	12	2	26	0	0	0.03	0.05	N/a	N/a	N/a	N/a	0	0	0	17.3	53	72.6
Armel Bell	GER	Southampt	DF	23-147	4	2	331	0	0	0	0	0.3	0	0	5	0	0	0	0.09	0	N/a	N/a	N/a	N/a	0	0	0	9.1	202	90.2
Ashley You	ENG	Everton	DF,FW	39-302	29	17	1622	1	3	6	1	0.3	1.6	20	83	27	0.06	0.17	0.01	0.09	N/a	N/a	N/a	N/a	50	0.17	0.17	24.5	610	69.9
Axel Disasi	FRA	Aston Villa	DF	27-057	7	5	490	0	0	2	0	0.1	0.4	10	16	5	0	0	0.02	0.07	N/a	N/a	N/a	N/a	0	0	0	19.5	243	86.8
Axel Disasi	FRA	Chelsea	DF	27-057	6	4	364	1	0	1	0	0.3	0.1	6	16	2	0.25	0	0.07	0.02	N/a	N/a	N/a	N/a	50	0.25	0.5	12.3	266	90.8
Axel Tuanz	COD	Ipswich To	DF	27-174	19	17	1446	0	1	5	1	0.2	0.3	9	46	8	0	0.06	0.01	0.02	N/a	N/a	N/a	N/a	0	0	0	15.7	538	83

Data characteristics:

- Players are sorted by name for easier lookup.
- All missing or inapplicable statistics are marked as "N/a".

- The data is de-duplicated and standardized into a consistent structure.

3. Statistical Data Analysis

3.1. Objective

Analyze player and team performance based on statistical data to:

- Evaluate individual and team performance.
- Identify the best-performing team based on multiple indicators.
- Visualize the distribution of key performance metrics.

3.2. Tools Used

- **Python**: primary programming language.
- **Pandas, NumPy**: for data handling and basic statistics.
- **Matplotlib, Seaborn**: for data visualization (histograms).
- **PdfPages**: to export multiple plots into a single PDF file.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import logging
import sys
import re
from pathlib import Path
from matplotlib.backends.backend_pdf import PdfPages
```

3.3. Process

a. Data Loading and Preprocessing:

- Load data from results.csv, replace "N/a" with NaN.

- Convert the Age column from the format "years-days" to float.

- Non-numeric columns (e.g., Player, Team, Nation) are excluded to focus on numerical analysis.

```
def load_data(self):
    try:
        self.logger.info("Loading data from results.csv...")
        self.df = pd.read_csv('results.csv')
        self.df.replace("N/a", np.nan, inplace=True)
        self.logger.info(f"Loaded {len(self.df)} rows of data")

        def convert_age(age_str):
            try:
                years, days = map(int, str(age_str).split('-'))
                return years + (days / 365)
            except:
                return pd.to_numeric(age_str, errors='coerce')

        self.df['Age'] = self.df['Age'].apply(convert_age).round(2)

        playing_time_cols = ['MP', 'Starts', 'Minutes']
        for col in playing_time_cols:
            self.df[col] = pd.to_numeric(self.df[col], errors='coerce')

        exclude_cols = ['Player', 'Nation', 'Team', 'Position']
        self.numeric_columns = self.df.select_dtypes(include=[np.number]).columns
        self.numeric_columns = [col for col in self.numeric_columns if col not in exclude_cols]

        self.logger.info(f"Found {len(self.numeric_columns)} numeric columns for analysis")
        self.logger.info(f"Columns: {', '.join(self.numeric_columns)}")

    except Exception as e:
        self.logger.error(f"Error loading data: {str(e)}")
        raise
```

b. Identify Top/Bottom 3 Players:

- For each metric, determine the **Top 3** and **Bottom 3** players.

- Results are saved in top_3.txt, helpful for highlighting outstanding or underperforming players.

```
def find_top_bottom_players(self):
    with open(self.output_dir / 'top_3.txt', 'w', encoding='utf-8') as f:
        all_stats = ['Age', 'MP', 'Starts', 'Minutes']
        all_stats.extend([col for col in self.numeric_columns if col not in all_stats])

        for stat in all_stats:
            f.write(f"\n{'='*50}\n")
            f.write(f"Statistic: {stat}\n")
            f.write(f"{'-'*50}\n")

            f.write("\nTop 3 Players:\n")
            top_3 = self.df.nlargest(3, stat)
            for _, row in top_3.iterrows():
                f.write(f"{row['Player']}: {row[stat]:.2f}\n")

            f.write("\nBottom 3 Players:\n")
            bottom_3 = self.df.nsmallest(3, stat)
            for _, row in bottom_3.iterrows():
                f.write(f"{row['Player']}: {row[stat]:.2f}\n")
```

c. Descriptive Statistics (Detailed)

This step involves computing three core statistical measures for each numeric metric in the dataset: **median**, **mean**, and **standard deviation**. These statistics help to summarize and understand the distribution and variability of player performance.

Metrics Calculated:

- **Mean (Average):**
  The mean is the average value, calculated by adding all values and dividing by the number of observations. It represents the overall tendency of the data.

- **Median:**
  The median is the middle value when the data is sorted. It is useful for understanding the central point of a dataset, especially when there are outliers.

- **Standard Deviation (Std):**
  Standard deviation shows how spread out the values are around the mean. A small standard deviation indicates that values are close to the mean, while a large one means there is a wide range of values.

Scope of Calculation:

- **Across the Entire Dataset:**
  Statistics are computed using data from all players, regardless of their team. This gives an overview of the league-wide performance distribution.

- **Per Team:**
  For each team, the mean, median, and standard deviation are calculated based on the players belonging to that team. This helps identify teams with consistent or standout performances in specific metrics.

- **Per Player:**
  Each player has a single row of data, so:

  - The **mean** and **median** are equal to the value of the metric itself.

- The **standard deviation** is zero, since there is no variation in a single value.

```
def calculate_statistics(self):
    try:
        team_stats = []
        player_stats = []

        all_stats = ['Age', 'MP', 'Starts', 'Minutes']
        all_stats.extend([col for col in self.numeric_columns if col not in all_stats])

        all_stats_dict = {'Team': 'all'}
        for stat in all_stats:
            all_stats_dict[f'Median of {stat}'] = self.df[stat].median()
            all_stats_dict[f'Mean of {stat}'] = self.df[stat].mean()
            all_stats_dict[f'Std of {stat}'] = self.df[stat].std()
        team_stats.append(all_stats_dict)

        for team in self.df['Team'].dropna().unique():
            team_data = self.df[self.df['Team'] == team]
            stats = {'Team': team}
            for stat in all_stats:
                stats[f'Median of {stat}'] = team_data[stat].median()
                stats[f'Mean of {stat}'] = team_data[stat].mean()
                stats[f'Std of {stat}'] = team_data[stat].std()
            team_stats.append(stats)

        all_player_stats = {'Player': 'all'}
        for stat in all_stats:
            all_player_stats[f'Median of {stat}'] = self.df[stat].median()
            all_player_stats[f'Mean of {stat}'] = self.df[stat].mean()
            all_player_stats[f'Std of {stat}'] = self.df[stat].std()
        player_stats.append(all_player_stats)

        for _, row in self.df.iterrows():
            stats = {'Player': row['Player']}
            for stat in all_stats:
                stats[f'Median of {stat}'] = row[stat]
                stats[f'Mean of {stat}'] = row[stat]
                stats[f'Std of {stat}'] = 0
            player_stats.append(stats)

        team_df = pd.DataFrame(team_stats).set_index('Team')
        player_df = pd.DataFrame(player_stats).set_index('Player')

        column_groups = []
        for stat in all_stats:
            column_groups.extend([f'Median of {stat}', f'Mean of {stat}', f'Std of {stat}'])

        team_df = team_df[column_groups]
        player_df = player_df[column_groups]
```

Output Format:

- The computed results are stored in the file results2.csv, which is structured into two main sections:
  - **Part 1: Team-level Statistics**
    Contains the mean, median, and standard deviation for each metric per team (and for the overall league).
  - **Part 2: Player-level Statistics**
    Lists the same statistics for each player.

d. Histogram Plotting:

- Only histograms for **6 representative metrics** are generated:
  - **Attacking**: Goals, Sh (Shots), Assists
  - **Defensive**: Tkl, Att, Blocks
- Each histogram is plotted:
  - For all players
  - For each team

- All plots are saved in all_distributions.pdf (ideal for presentation and printing).

```
def plot_distributions(self):
    try:
        sns.set(style="whitegrid")


        attack_stats = ['Goals', 'Sh', 'Assists']
        defense_stats = ['Tkl', 'Att', 'Blocks']
        all_stats = attack_stats + defense_stats

        pdf_path = self.plots_dir / 'all_distributions.pdf'
        with PdfPages(pdf_path) as pdf:
            for stat in all_stats:
                if stat not in self.df.columns:
                    print(f"[SKIP] {stat} - not found in dataframe.")
                    continue
                if not pd.api.types.is_numeric_dtype(self.df[stat]):
                    print(f"[SKIP] {stat} - not numeric.")
                    continue

                plt.figure(figsize=(10, 5))
                sns.histplot(data=self.df, x=stat, bins=30, kde=True, color="blue")
                plt.title(f"Distribution of {stat} - All Players")
                plt.xlabel(stat)
                plt.ylabel("Frequency")
                plt.tight_layout()
                pdf.savefig()
                plt.close()

                for team in self.df['Team'].dropna().unique():
                    plt.figure(figsize=(10, 5))
                    team_data = self.df[self.df['Team'] == team]
                    sns.histplot(data=team_data, x=stat, bins=20, kde=True, color="green")
                    plt.title(f"Distribution of {stat} - {team}")
                    plt.xlabel(stat)
                    plt.ylabel("Frequency")
                    plt.tight_layout()
                    pdf.savefig()
                    plt.close()
```

e. Determine the Best Team:

- For each metric, identify the team with the highest average value.

- Count the number of metrics each team leads in → the team with the highest count is considered the **best-performing team**.

- Results are written to team_results.txt, including:

    - List of top-performing teams per metric.

○ The team that leads in the most metrics.

3.4. Outputs

- **top_3.txt**:

    ○ Lists of Top 3 and Bottom 3 players for each metric.

    ○ Useful for highlighting star players and identifying underperformers.

```
==================================================
Statistic: MP
--------------------------------------------------

Top 3 Players:
Alex Iwobi: 35.00
Anthony Elanga: 35.00
Bernd Leno: 35.00

Bottom 3 Players:
Altay Bayındır: 2.00
Ayden Heaven: 2.00
Billy Gilmour: 2.00

==================================================
Statistic: Starts
--------------------------------------------------

Top 3 Players:
Bernd Leno: 35.00
Bruno Guimarães: 35.00
Bryan Mbeumo: 35.00
```

- **results2.csv**:

    ○ Descriptive statistics (median, mean, standard deviation) for:

        ■ The entire league

        ■ Individual teams

■ Individual players

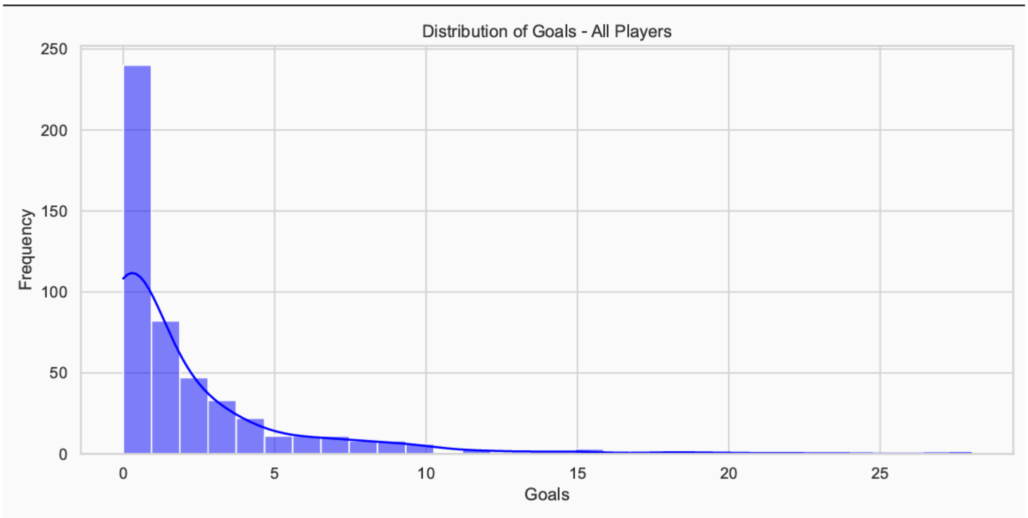
Team Statistics:

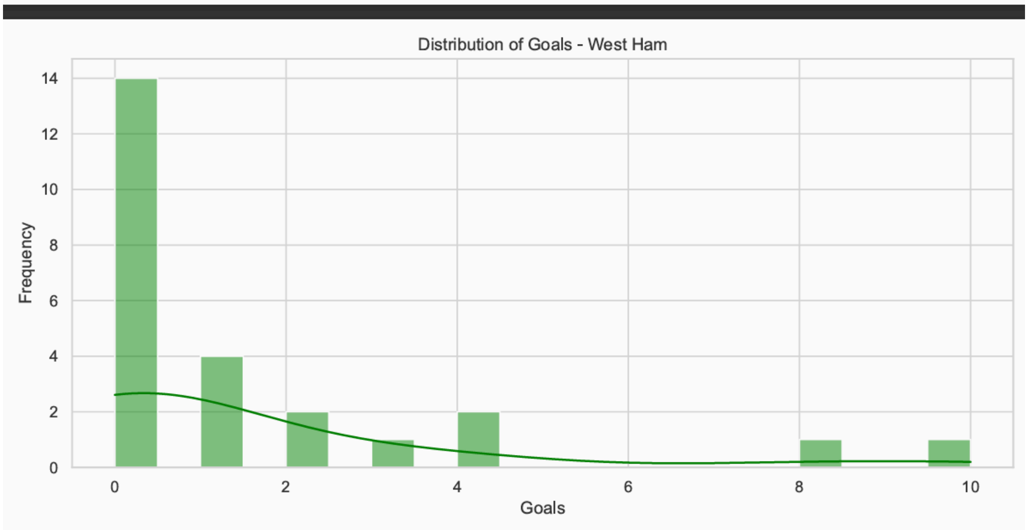
Team	Median of	Mean of A	Std of Age	Median of	Mean of M	Std of MP	Median of	Mean of S	Std of Star	Median of	Mean of M	Std of Min	Median of	Mean of G	Std of Goa	Median of	Mean of A	Std of Assi	Median of	Mean of Y	Std of Yell	Median of	Mean of R	Std of Red	Median of	Mean of x	Std of xG	Median of	Mean of x	Std of xAG	M
all	26.58	26.87968	4.250709	23	21.22267	9.975986	14	15.5668	10.95716	1337	1395.866	927.799	1	2.030364	3.548718	1	1.508097	2.219893	2	2.995951	2.636147	0	0.091093	0.308437	0.9	2.04919	3.123618	0.9	1.516194	1.890614	
West Ham	28.34	28.622	5.01522	20	21.56	8.529752	14	15.4	10.86278	1070	1381.52	914.3885	0	1.48	2.6	1	0.96	1.513275	2	2.96	2.6533	0	0.12	0.331662	1.2	1.776	2.254861	1.1	1.26	1.403567	
Southampt	26.98	26.97897	4.225519	20	18.65517	10.3795	13	13.24138	9.840351	1178	1184.31	856.2373	0	0.827586	1.071346	0	0.517241	0.828971	2	2.862069	2.948658	0	0.103448	0.409253	0.5	1.072414	1.387932	0.5	0.810345	0.922088	
Everton	26.54	27.97864	5.032906	24	22.40909	9.287263	15	17.45455	10.64907	1357.5	1565.227	904.2692	1	1.5	2.132515	1	1	1.112697	3	3.409091	2.174727	0	0.090909	0.294245	1.1	1.7	2.018958	1.05	1.3	1.10798	
Manchest	26.69	26.9996	4.913701	23	19.88	8.973665	17	15.36	8.62593	1494	1381.36	766.2823	1	2.64	4.414748	0	1.92	2.564501	2	2.32	1.749286	0	0.04	0.2	1.3	2.52	4.175424	1.5	2.028	2.249355	
Leicester (	26.755	27.22154	4.462351	21.5	20.23077	9.717233	15	14.80769	9.968026	1408	1329.192	824.5063	0	1.115385	2.006528	0	0.846154	1.189699	2	3.115385	2.643133	0	0	0	0.4	1.184615	2.171579	0.4	0.919231	1.140533	
Bournemo	26.01	26.47522	3.839318	26	22.21739	10.18	17	16.69565	11.63238	1670	1494.609	1004.394	1	2.347826	3.587968	1	1.695652	2.054591	3	3.869565	3.07932	0	0.086957	0.288104	1.1	2.682609	3.577156	1.1	1.808696	1.874206	
Brighton	24.53	26.04286	5.171661	21	19.53571	10.03506	10	13.75	10.12743	929.5	1233.821	890.9927	1	1.964286	2.99978	1	1.321429	1.611385	2	2.678571	2.597058	0	0.107143	0.31497	0.75	1.910714	2.643558	1	1.321429	1.398506	
Crystal Pal	27.19	27.17952	3.134588	30	24.04762	10.63239	19	18.2381	12.87208	1652	1633.048	1082.85	0	1.904762	3.404479	1	1.52381	2.088517	2	3.619048	3.368623	0	0.190476	0.402374	1.1	2.552381	3.463902	1.5	1.914286	2.128212	
Fulham	28.83	28.82727	3.350788	27	24.5	9.328655	17.5	17.45455	11.49628	1620.5	1568.273	963.2437	0.5	2.227273	3.264992	1	1.909091	2.56179	2.5	3.409091	2.986622	0	0.090909	0.294245	1.05	2.122727	2.684866	1	1.627273	1.882294	
Manchest	25.725	25.48467	5.089674	20	17.66667	11.45406	11.5	12.83333	10.7802	1071	1149.667	931.0153	0	1.333333	2.264164	0	0.933333	1.99885	2	2.6	2.471911	0	0.1	0.402578	0.7	1.603333	2.267687	0.35	1.2	1.70759	
Ipswich To	26.705	26.90733	3.155321	18.5	18.2	9.01493	11.5	12.83333	9.684981	984.5	1146.767	798.9913	0	1.133333	2.315366	0	0.833333	1.085431	2	2.966667	2.645534	0	0.166667	0.379049	0.6	1.07	1.759732	0.4	0.71	0.971437	
Newcastle	27.47	27.96652	4.724641	27	23.21739	10.14422	14	16.73913	12.58866	1503	1502.87	1050.911	0	2.782609	5.187016	1	2.086957	2.843062	2	2.608696	3.026236	0	0.043478	0.208514	0.5	2.634783	4.436994	0.9	1.882609	2.277489	
Liverpool	26.45	27.2181	3.688814	28	25.19048	8.925352	20	18.33333	11.92197	1717	1643.095	977.499	1	3.809524	6.49322	2	2.857143	3.991061	3	3.047619	2.268836	0	0.095238	0.300793	1.7	3.67619	5.513974	1.2	2.67619	3.184636	
Aston Villa	27.64	27.145	4.037085	20.5	19.42857	10.33308	9.5	13.75	11.66071	1004	1235.357	943.1826	1	1.892857	3.292407	0	1.5	2.395984	2	2.428571	2.251396	0	0.071429	0.262265	0.8	1.953571	2.972526	0.45	1.432143	2.008326	
Wolves	26.86	27.6813	3.99398	26	22.78261	8.923699	15	16.65217	10.90708	1364	1495.217	875.6225	1	2.173913	3.93876	1	1.695652	2.224549	2	3.217391	2.74618	0	0.086957	0.288104	0.8	1.76087	2.479872	0.9	1.4	1.779428	
Nott'ham	27.145	27.27909	3.592831	30	24.5	10.88796	19	17.5	13.30145	1804	1572.682	1104.925	1	2.409091	4.159431	1	1.772727	2.486761	3	3.727273	3.239368	0	0.090909	0.294245	1.4	1.922727	2.604745	1.1	1.409091	1.431918	
Tottenhan	25.65	25.66556	4.538351	22	19.33333	9.232051	16	14.22222	8.092415	1300	1277.778	694.5651	1	2.222222	3.250247	1	1.703704	2.382869	2	2.444444	2.325996	0	0.037037	0.19245	0.7	2.1	2.909137	0.9	1.581481	2.020098	
Chelsea	24.315	24.21077	2.377928	18	19.69231	11.26328	11.5	14.80769	11.79159	1046.5	1329.5	1029.045	1	2.269231	3.65029	1	1.730769	2.254909	2.5	3.692308	3.197114	0	0.038462	0.196116	0.75	2.565385	4.209508	1.05	1.965385	2.476278	
Brentford	24.84	26.20381	3.914391	28	23.52381	11.38253	21	18.33333	13.29787	1913	1638.095	1122.036	0	2.904762	5.539899	2	1.952381	2.438774	2	2.52381	1.887301	0	0.047619	0.218218	0.8	2.609524	4.196773	1.2	1.895238	2.231026	
Arsenal	26.885	26.50318	3.822545	23.5	23.27273	8.066284	16.5	17.5	10.39574	1478.5	1564.455	902.9765	2	2.818182	2.805375	1.5	2.318182	2.749656	2.5	2.909091	1.973855	0	0.227273	0.528413	1.65	2.563636	2.640986	0.9	1.931818	2.187712	

Player Statistics:

Player	Median of	Mean of A	Std of Age	Median of	Mean of M	Std of MP	Median of	Mean of S	Std of Star	Median of	Mean of M	Std of Min	Median of	Mean of G	Std of Goa	Median of	Mean of A	Std of Assi	Median of	Mean of Y	Std of Yell	Median of	Mean of R	Std of Red	Median of	Mean of x	Std of xG	Median of	Mean of x	Std of xAG	M
all	26.58	26.87968	4.250709	23	21.22267	9.975986	14	15.5668	10.95716	1337	1395.866	927.799	1	2.030364	3.548718	1	1.508097	2.219893	2	2.995951	2.636147	0	0.091093	0.308437	0.9	2.04919	3.123618	0.9	1.516194	1.890614	
Aaron Cres	35.39	35.39	0	15	15	0	8	8	0	676	676	0	0	0	0	0	0	0	2	2	0	0	0	0	0.1	0.1	0	1.1	1.1	0	
Aaron Ran	26.98	26.98	0	27	27	0	27	27	0	2430	2430	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	
Aaron Wai	27.44	27.44	0	33	33	0	32	32	0	2884	2884	0	2	2	0	3	3	0	1	1	0	0	0	0	1.2	1.2	0	3.2	3.2	0	
Abdoulaye	32.35	32.35	0	30	30	0	29	29	0	2425	2425	0	3	3	0	1	1	0	5	5	0	1	1	0	3.9	3.9	0	2.3	2.3	0	
Abdukodir	21.18	21.18	0	6	6	0	6	6	0	503	503	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0.1	0.1	0	
Abdul Fata	21.16	21.16	0	11	11	0	6	6	0	579	579	0	0	0	0	2	2	0	0	0	0	0	0	0	0.4	0.4	0	1.6	1.6	0	
Adam Arm	28.24	28.24	0	20	20	0	15	15	0	1248	1248	0	2	2	0	2	2	0	4	4	0	0	0	0	3.3	3.3	0	1.2	1.2	0	
Adam Lalla	36.99	36.99	0	14	14	0	5	5	0	361	361	0	0	0	0	2	2	0	4	4	0	0	0	0	0.2	0.2	0	0.9	0.9	0	
Adam Smit	34.02	34.02	0	22	22	0	17	17	0	1409	1409	0	0	0	0	0	0	0	6	6	0	0	0	0	0.7	0.7	0	0.3	0.3	0	
Adam Wel	30.34	30.34	0	11	11	0	8	8	0	617	617	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	
Adam Wha	20.93	20.93	0	20	20	0	16	16	0	1318	1318	0	0	0	0	2	2	0	2	2	0	0	0	0	0.4	0.4	0	3	3	0	
Adama Tra	29.28	29.28	0	33	33	0	16	16	0	1592	1592	0	2	2	0	6	6	0	3	3	0	0	0	0	3.9	3.9	0	4.7	4.7	0	
Albert GrÃ	23.96	23.96	0	4	4	0	2	2	0	143	143	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.1	0	0	0	0	
Alejandro	20.85	20.85	0	34	34	0	23	23	0	2146	2146	0	6	6	0	2	2	0	3	3	0	0	0	0	7.2	7.2	0	4.5	4.5	0	
Alex Iwobi	29.01	29.01	0	35	35	0	33	33	0	2796	2796	0	9	9	0	6	6	0	1	1	0	0	0	0	4.6	4.6	0	6.9	6.9	0	
Alex McCa	35.42	35.42	0	5	5	0	5	5	0	450	450	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Alex Palme	28.74	28.74	0	11	11	0	11	11	0	990	990	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	
Alex Scott	21.71	21.71	0	18	18	0	7	7	0	709	709	0	0	0	0	0	0	0	2	2	0	0	0	0	0.7	0.7	0	0.8	0.8	0	
Alexander	25.62	25.62	0	32	32	0	32	32	0	2577	2577	0	23	23	0	6	6	0	1	1	0	0	0	0	19.9	19.9	0	4.3	4.3	0	

- **all_distributions.pdf**:
  - Contains histograms for 6 key metrics (Goals, Sh, Assists, Tkl, Att, Blocks)
  - Each with global and team-specific views.





- **team_results.txt**:
    - List of teams that lead each metric.
    - Highlights the team that dominates across the most performance indicators.

```
Best Team by Each Statistic:
- Age: Fulham (28.83)
- MP: Liverpool (25.19)
- Starts: Brentford (18.33)
- Minutes: Liverpool (1643.10)
- Goals: Liverpool (3.81)
- Assists: Liverpool (2.86)
- Yellow Cards: Bournemouth (3.87)
- Red Cards: Arsenal (0.23)
- xG: Liverpool (3.68)
- xAG: Liverpool (2.68)
- PrgC: Manchester City (41.64)
- PrgP: Liverpool (83.95)
- PrgR: Liverpool (83.05)
- Gls/90: Manchester City (0.18)
- Ast/90: Liverpool (0.14)
- xG/90: Aston Villa (0.19)
- xGA/90: Chelsea (0.15)
- SoT/90: Bournemouth (0.54)
- Cmp: Liverpool (807.14)
- Cmp%: Manchester City (86.50)
- TotDist: Liverpool (4594.33)
- Short%: Manchester City (92.13)
- Med%: Manchester City (89.52)
- KP: Liverpool (22.43)
- Passes 1/3: Liverpool (69.57)
```

4. Clustering Players using K-means and PCA

4.1. Objective

To classify Premier League players into distinct clusters based on their playing style or performance using the **K-means clustering** algorithm combined with **Principal Component Analysis (PCA)** for dimensionality reduction. This aims to:

- Better understand the distribution of players based on their playing characteristics.

- Identify groups of similar players (e.g., creative midfielders, strong tackling defenders, shot-stopping goalkeepers, etc.).

4.2. Tools Used

- **Python**

- **scikit-learn**: for implementing KMeans, PCA, and calculating silhouette_score.

- **Pandas**, **NumPy**: for data processing and normalization.

- **Matplotlib**: for visualizing clustering results.

- **PdfPages**: for exporting plots to a file named classify.pdf.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import logging
```

4.3. Method Summary

a. Selecting representative metrics:

A set of metrics reflecting general performance, suitable for both outfield players and goalkeepers:

- **Attacking**: Goals, xG, GCA, SCA, Assists
- **Shooting**: SoT%, SoT/90, G/Sh
- **Passing and progression**: Cmp%, KP, PrgP, PrgC, Passes 1/3, PPA, CrsPA
- **Defending**: Tkl, TklW, Int, Blocks, Sh
- **Goalkeeping**: Save%, CS%, GA90, PKsv%
- **General**: Touches, Lost, Won%, Recov

b. Data normalization:

- Replace "N/a" values with NaN, then fill missing values with 0.

- Normalize the data using StandardScaler to bring all features onto the same scale (mean = 0, standard deviation = 1).

```
# Load and prepare data with mixed features for all roles including goalkeepers
def load_and_prepare_data(file_path='results.csv'):
    df = pd.read_csv(file_path)
    df.rename(columns=lambda x: x.strip(), inplace=True)

    # Selected hybrid feature set
    selected_columns = [
    'Goals', 'xG', 'GCA', 'SCA', 'Assists',
    'SoT%', 'SoT/90', 'G/Sh',
    'Cmp%', 'KP', 'PrgP', 'PrgC',
    'Passes 1/3', 'PPA', 'CrsPA',
    'Tkl', 'TklW', 'Int', 'Blocks', 'Sh',
    'Save%', 'CS%', 'GA90', 'PKsv%',
    'Touches', 'Lost', 'Won%', 'Recov'
]

    # Handle missing data: fill goalkeeper stats with 0 for outfield players and vice versa
    df_features = df[selected_columns].replace("N/a", np.nan).fillna(0)
    df_features = df_features.astype(float)

    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df_features)

    return df, df_features, scaled_features
```

c. Determining the optimal number of clusters:

- Run KMeans for k values ranging from 2 to 9.

- Compute the silhouette score for each k to assess how well-defined the clusters are.

- Select the optimal k as the one with the highest silhouette score.

```
# Determine best number of clusters
def determine_best_k(data, k_range=range(2, 10)):
    scores = []
    valid_k = []
    for k in k_range:
        try:
            kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
            labels = kmeans.fit_predict(data)
            if len(set(labels)) <= 1:
                continue
            score = silhouette_score(data, labels)
            scores.append(score)
            valid_k.append(k)
        except Exception as e:
            logging.warning(f"Silhouette score failed for k={k}: {e}")
            continue

    if not scores:
        raise ValueError("No valid k values found for silhouette score.")

    best_k = valid_k[np.argmax(scores)]
    return best_k, scores
```

d. Dimensionality reduction using PCA:

- Apply PCA to reduce the number of dimensions to 2 for visualization purposes.

- Retain most of the data variance in the two main components: PCA1 and PCA2.

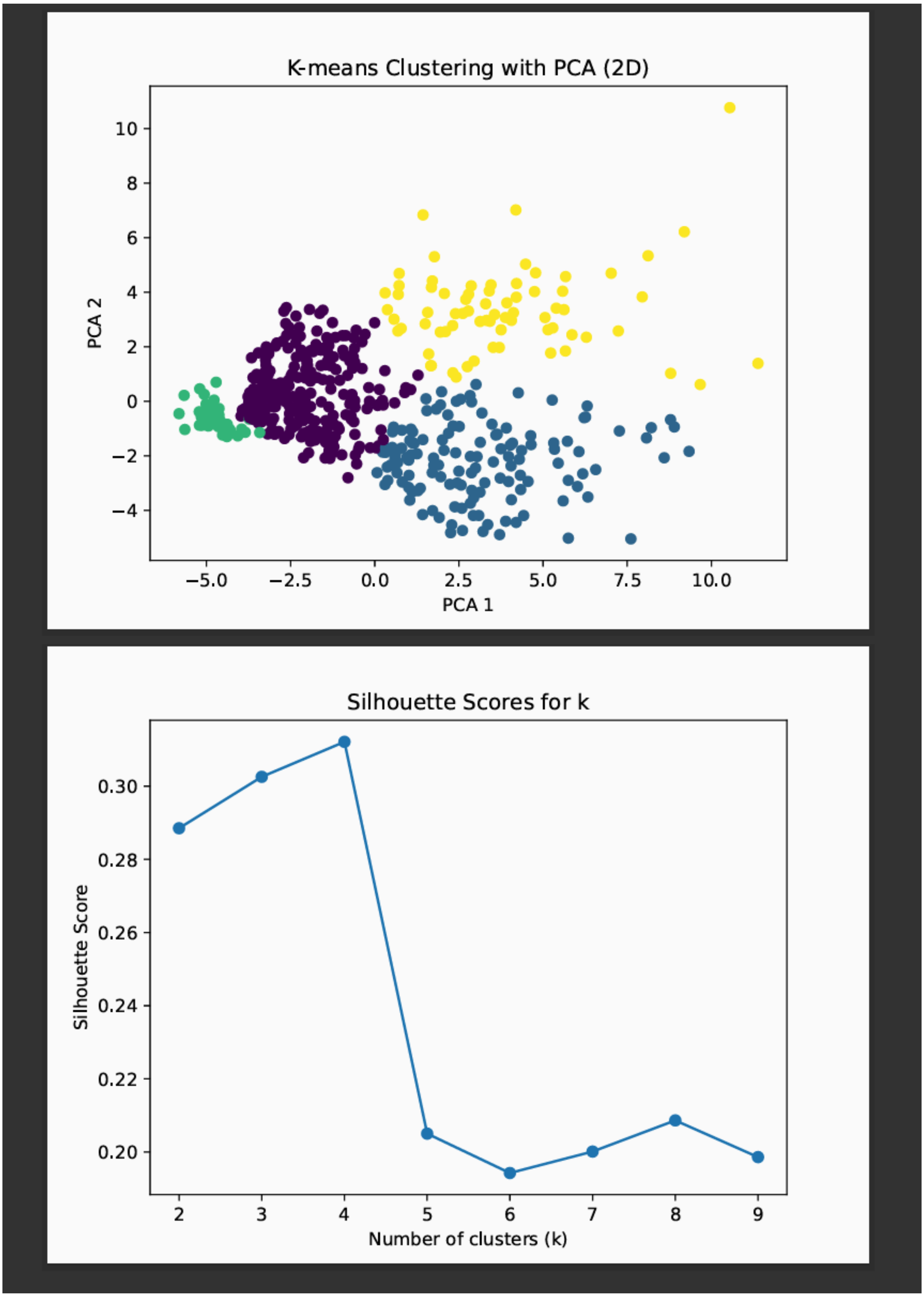
```
# Plot PCA clusters
def plot_clusters(data, labels):
    pca = PCA(n_components=2)
    reduced = pca.fit_transform(data)
    fig = plt.figure()
    plt.scatter(reduced[:, 0], reduced[:, 1], c=labels, cmap='viridis', s=30)
    plt.title('K-means Clustering with PCA (2D)')
    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    return fig
```

e. Plotting the results:

- A 2D scatter plot to visualize the PCA-reduced clusters.
- A line plot showing silhouette scores for each value of k.

f. Saving the results:

- Export both plots into a PDF file named classify.pdf for easy sharing or presentation.



4.4. Results

- The optimal number of clusters identified was **4**, suggesting the existence of multiple groups of players with distinct styles and performance profiles.

- **PCA 2D cluster plot**: clusters are well-separated with distinct colors, making them easy to identify.

- **Silhouette score plot**: clearly supports the choice of $k = 4$ as a reasonable and data-backed decision.

5. Predicting Player Transfer Values

5.1. Objective

To estimate the transfer values of Premier League players based on:

- Performance metrics
- Personal attributes
- Experience and playing position

This allows for data-driven suggestions on which players should be bought or sold to optimize investment efficiency.

5.2. Tools Used

- **Python**
- **Selenium**: For scraping data from footballtransfers.com
- **Pandas, NumPy**: For data manipulation and preprocessing
- **scikit-learn**: For machine learning (using RandomForestRegressor, LabelEncoder, train_test_split, and metrics)
- **fuzzywuzzy**: For fuzzy matching player names between data sources
- **seaborn, matplotlib, PdfPages**: For visualization

```
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
from selenium.common.exceptions import TimeoutException, StaleElementReferenceException, NoSuchElementException
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from fuzzywuzzy import fuzz
from unidecode import unidecode
import time
import os
import tempfile
```

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

5.3. Process Summary

a. Filter input data:

- From results.csv, select only players who played more than 900 minutes to focus on those with enough data for analysis.

b. Scrape market values:

- Access 22 pages on FootballTransfers to collect Player, Age, Team, and Transfer_Value.
- Automatically handle cookies, page load errors, and StaleElementReference issues.

```python
table = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "table.table")))
rows = table.find_elements(By.TAG_NAME, "tr")[1:]

page_data = []
for row in rows:
    try:
        cols = row.find_elements(By.TAG_NAME, "td")
        if len(cols) >= 6:
            player_data = {
                'Player': cols[2].text.strip(),
                'Team': cols[4].text.strip(),
                'Transfer_Value': cols[5].text.strip()
            }
            page_data.append(player_data)
    except StaleElementReferenceException:
        continue

if page_data:
    all_players.extend(page_data)
    print(f"  → Đã thu thập {len(page_data)} cầu thủ từ trang {page_num}")
```

c. Merge with FBref data:

- Use fuzzy name matching to align player names between FBref and FootballTransfers.
- The result is transfer_values.csv containing: Player, Team, Position, Age, Minutes, and Transfer_Value.

```python
def match_and_filter_data(results_df, transfer_df, similarity_threshold=75):
    print("\nBước 3: Đang so khớp dữ liệu...")
    matched_data = []

    for _, player in results_df.iterrows():
        best_match = None
        best_score = 0
        player_name_norm = normalize_name(player['Player'])

        for _, transfer in transfer_df.iterrows():
            transfer_name_norm = normalize_name(transfer['Player'])
            score = fuzz.token_set_ratio(player_name_norm, transfer_name_norm)
            if score > best_score:
                best_score = score
                best_match = transfer

        if best_score >= similarity_threshold:
            matched_data.append({
                'Player': player['Player'],
                'Team': player['Team'],
                'Minutes': player['Minutes'],
                'Position': player['Position'],
                'Age': player['Age'],
                'Transfer_Value': best_match['Transfer_Value']
            })
            print(f"  + Khớp: {player['Player']} ({best_score}%)")

    matched_df = pd.DataFrame(matched_data)
    print(f"- Đã tìm thấy {len(matched_df)} cặp khớp với độ tương đồng >= {similarity_threshold}%")
    return matched_df
```

d. Create model input features:

- Minutes_Normalized: Normalize playing time.
- Age_Group: Categorize players by age (<21, 21–25, 26–30, 30+).
- Experience_Level: Group by minutes played.
- Position_Type: Categorize by position (GK, DEF, MID, FWD).

- Encode categorical features using LabelEncoder.

```python
# Tiền xử lý dữ liệu
def preprocess_data(df):
    print("\nThông tin dữ liệu trước khi xử lý:")
    print(df.info())
    print("\nMẫu dữ liệu ban đầu:")
    print(df.head())

    # Chuyển đổi Transfer_Value từ chuỗi sang float
    df['Transfer_Value'] = df['Transfer_Value'].str.replace('€', '').str.replace('M', '').astype(float)
    print(f"\nSố lượng dữ liệu sau khi chuyển đổi Transfer_Value: {len(df)}")

    # Nhóm tuổi
    df['Age_Group'] = pd.cut(df['Age'], bins=[0, 21, 25, 30, 100], labels=['<21', '21-25', '26-30', '30+'])

    # Chuẩn hóa Minutes
    scaler = StandardScaler()
    df['Minutes_Normalized'] = scaler.fit_transform(df[['Minutes']])

    # Kinh nghiệm dựa theo số phút thi đấu
    df['Experience_Level'] = pd.cut(df['Minutes'], bins=[0, 1000, 2000, 3000, np.inf], labels=['Low', 'Medium', 'High', 'Very High'])

    # Phân loại Position
    position_map = {
        'GK': 'GK',
        'DF': 'DEF',
        'MF': 'MID',
        'FW': 'FWD'
    }
    df['Position_Type'] = df['Position'].map(lambda x: position_map.get(x[:2], 'Other'))

    print(f"\nSố lượng dữ liệu sau khi tạo Position_Type: {len(df)}")

    print("\nSố lượng giá trị NaN trong từng cột:")
    print(df.isnull().sum())

    df = df.dropna()
    print(f"\nSố lượng dữ liệu sau khi xóa các hàng có NaN: {len(df)}")

    print("\nThông tin dữ liệu sau khi xử lý:")
    print(df.info())
    print("\nMẫu dữ liệu sau khi xử lý:")
    print(df[['Player', 'Age', 'Position_Type', 'Transfer_Value']].head())

    return df
```

e. Train the Random Forest model:

- Train a RandomForestRegressor to predict Transfer_Value.

```python
# Hàm đánh giá mô hình
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    print(f"\nRMSE: {rmse:.2f}M")
    print(f"R2 Score: {r2:.2f}")

    return y_test, y_pred, rmse

# Hàm khuyến nghị chuyển nhượng
def recommend_transfers(df, y_pred):
    df_result = df.copy()
    df_result['Predicted_Value'] = y_pred
    df_result['Difference'] = df_result['Predicted_Value'] - df_result['Transfer_Value']

    print("\nTop 10 cầu thủ nên mua (giá trị dự đoán cao hơn giá thị trường):")
    print(df_result.sort_values(by='Difference', ascending=False).head(10)[['Player', 'Team', 'Transfer_Value', 'Predicted_Value', 'Difference']])

    print("\nTop 10 cầu thủ nên bán (giá trị dự đoán thấp hơn giá thị trường):")
    print(df_result.sort_values(by='Difference').head(10)[['Player', 'Team', 'Transfer_Value', 'Predicted_Value', 'Difference']])

    # Lưu ra file nếu cần
    df_result.to_csv('player_transfer_recommendations.csv', index=False)
```

```python
# Huấn luyện mô hình
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Đánh giá mô hình
y_test, y_pred, rmse = evaluate_model(model, X_test, y_test)

# Dự đoán toàn bộ cầu thủ
y_all_pred = model.predict(X)
```

f. Evaluate the model:

- Use RMSE (Root Mean Squared Error) and R^2 score to assess accuracy.
- Predict values for all players and save to player_transfer_recommendations.csv.

g. Generate transfer recommendations:

- Calculate the difference between predicted and actual market values.
- Recommendations:
  - **Buy**: Players where Predicted_Value > Transfer_Value
  - **Sell**: Players where Predicted_Value < Transfer_Value

h. Visualize results:

- Scatter plot showing correlation between predicted and actual values.
- Bar chart showing feature importance.
- Export all results to player_value_report.pdf.

5.4. Results Obtained

- **transfer_values.csv**: List of successfully matched players with more than 900 minutes played and known market values.Collected data for 300 out of 304 players whose playing time is greater than 900 minutes

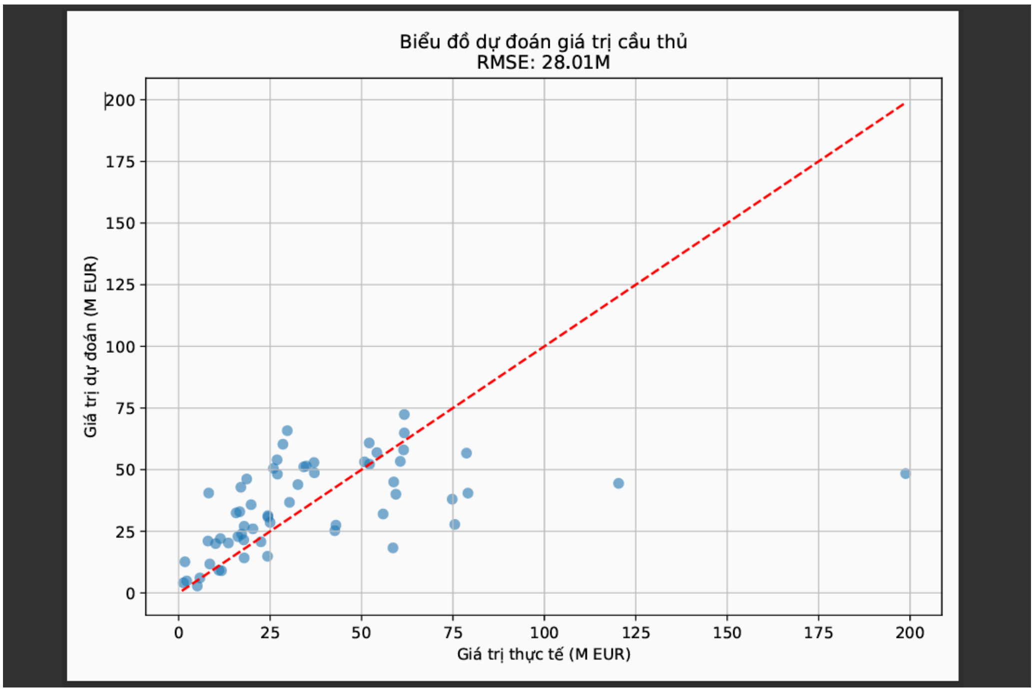
- **player_transfer_recommendations.csv**: Contains Predicted_Value and Difference, enabling actionable buy/sell suggestions.

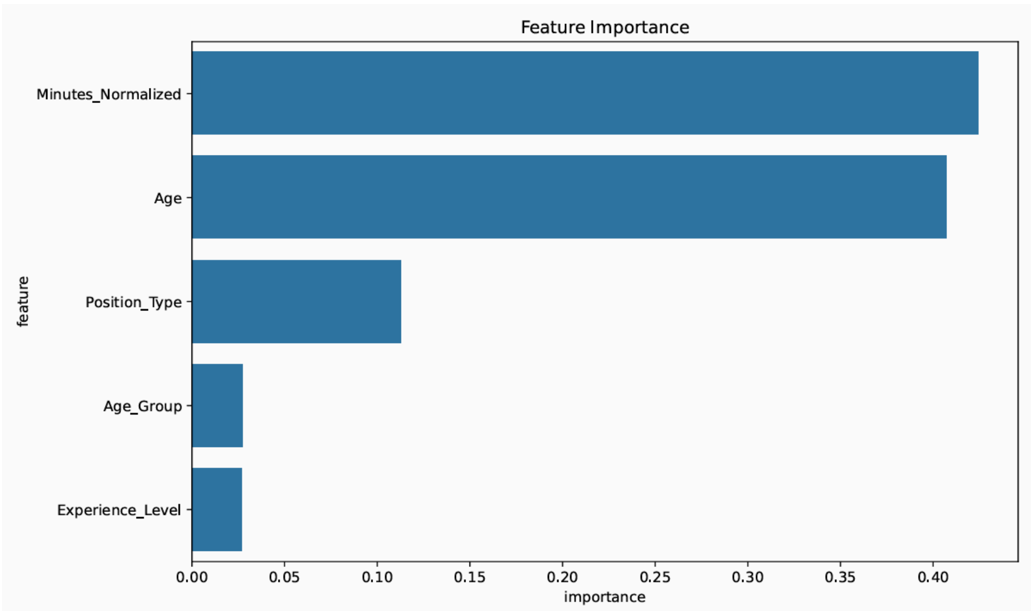
transfer_values.csv

```
Player,Team,Minutes,Position,Age,Transfer_Value
Aaron Ramsdale,Southampton,2430,GK,26,€18.7M
Aaron Wan-Bissaka,West Ham,2884,DF,27,€26.9M
Abdoulaye Doucouré,Everton,2425,MF,32,€5.8M
Adam Smith,Bournemouth,1409,DF,34,€1.5M
Adam Wharton,Crystal Palace,1318,MF,20,€48.9M
Adama Traoré,Fulham,1592,"FW,MF",29,€8M
Alejandro Garnacho,Manchester Utd,2146,"MF,FW",20,€60.7M
Alex Iwobi,Fulham,2796,"FW,MF",29,€30.3M
Alex Palmer,Ipswich Town,990,GK,28,€1.6M
Alexander Isak,Newcastle Utd,2577,FW,25,€120.3M
Alexis Mac Allister,Liverpool,2575,MF,26,€106.1M
Alisson,Liverpool,2238,GK,32,€24.1M
Alphonse Areola,West Ham,2080,GK,32,€11.8M
Amad Diallo,Manchester Utd,1639,"FW,MF",22,€48.5M
Amadou Onana,Aston Villa,1378,MF,23,€62.1M
Andreas Pereira,Fulham,1879,MF,29,€19M
Andrew Robertson,Liverpool,2308,DF,31,€24M
André,Wolves,2249,MF,23,€34.2M
André Onana,Manchester Utd,2970,GK,29,€34.2M
Anthony Elanga,Nott'ham Forest,2232,"FW,MF",23,€45.5M
Anthony Gordon,Newcastle Utd,2235,FW,24,€49.5M
Antoine Semenyo,Bournemouth,2933,FW,25,€47.7M
Antonee Robinson,Fulham,2989,DF,27,€46.4M
Archie Gray,Tottenham,1481,"DF,MF",19,€58.2M
Arijanet Muric,Ipswich Town,1620,GK,26,€10.1M
Ashley Young,Everton,1622,"DF,FW",39,€1.3M
Axel Tuanzebe,Ipswich Town,1446,DF,27,€4.2M
Bart Verbruggen,Brighton,2970,GK,22,€30.7M
Ben Davies,Tottenham,1126,DF,32,€5.6M
Ben Johnson,Ipswich Town,1348,"DF,FW",25,€12.6M
Ben White,Arsenal,941,DF,27,€58.8M
```

- **player_value_report.pdf**:
  - Scatter plot: Actual vs. predicted values

○ Feature importance bar chart: Shows which features most influence player value





1	Player	Team	Minutes	Position	Age	Transfer_V	Age_Grou	Minutes_N	Experience	Position_T	Predicted_	Difference
2	Aaron Ran	Southampt	2430	GK	26	18.7	26-30	0.65557	High	GK	30.218	11.518
3	Aaron Wai	West Ham	2884	DF	27	26.9	26-30	1.365297	High	DEF	28.722	1.822
4	Abdoulaye	Everton	2425	MF	32	5.8	30+	0.647753	High	MID	9.847	4.047
5	Adam Smit	Bournemo	1409	DF	34	1.5	30+	-0.94053	Medium	DEF	2.484	0.984
6	Adam Wha	Crystal Pal	1318	MF	20	48.9	<21	-1.08279	Medium	MID	46.56	-2.34
7	Adama Tra	Fulham	1592	FW,MF	29	8	26-30	-0.65445	Medium	FWD	21.065	13.065
8	Alejandro	Mancheste	2146	MF,FW	20	60.7	<21	0.2116	High	MID	55.942	-4.758
9	Alex Iwobi	Fulham	2796	FW,MF	29	30.3	26-30	1.227728	High	FWD	36.754	6.454
10	Alex Palme	Ipswich To	990	GK	28	1.6	26-30	-1.59555	Low	GK	10.639	9.039
11	Alexander	Newcastle	2577	FW	25	120.3	21-25	0.885371	High	FWD	44.449	-75.851
12	Alexis Mac	Liverpool	2575	MF	26	106.1	26-30	0.882245	High	MID	87.674	-18.426
13	Alisson	Liverpool	2238	GK	32	24.1	30+	0.355421	High	GK	19.764	-4.336
14	Alphonse	West Ham	2080	GK	32	11.8	30+	0.108424	High	GK	15.097	3.297
15	Amad Diall	Mancheste	1639	FW,MF	22	48.5	21-25	-0.58098	Medium	FWD	54.802	6.302
16	Amadou O	Aston Villa	1378	MF	23	62.1	21-25	-0.989	Medium	MID	55.227	-6.873
17	Andreas Pe	Fulham	1879	MF	29	19	26-30	-0.20579	Medium	MID	17.913	-1.087
18	Andrew Ro	Liverpool	2308	DF	31	24	30+	0.46485	High	DEF	20.085	-3.915
19	AndrÃ©	Wolves	2249	MF	23	34.2	21-25	0.372617	High	MID	51.145	16.945
20	AndrÃ© O	Mancheste	2970	GK	29	34.2	26-30	1.499738	High	GK	33.596	-0.604
21	Anthony El	Nott'ham	2232	FW,MF	23	45.5	21-25	0.346041	High	FWD	49.431	3.931
22	Anthony G	Newcastle	2235	FW	24	49.5	21-25	0.350731	High	FWD	45.934	-3.566
23	Antoine Se	Bournemo	2933	FW	25	47.7	21-25	1.441897	High	FWD	47.564	-0.136
24	Antonee R	Fulham	2989	DF	27	46.4	26-30	1.529441	High	DEF	41.558	-4.842
25	Archie Gra	Tottenham	1481	DF,MF	19	58.2	<21	-0.82798	Medium	DEF	51.829	-6.371
26	Arijanet M	Ipswich To	1620	GK	26	10.1	26-30	-0.61068	Medium	GK	20.037	9.937
27	Ashley You	Everton	1622	DF,FW	39	1.3	30+	-0.60756	Medium	DEF	4.116	2.816
28	Axel Tuanz	Ipswich To	1446	DF	27	4.2	26-30	-0.88269	Medium	DEF	19.35	15.15
29	Bart Verbr	Brighton	2970	GK	22	30.7	21-25	1.499738	High	GK	44.375	13.675
30	Ben Davies	Tottenham	1126	DF	32	5.6	30+	-1.38294	Medium	DEF	5.59	-0.01
31	Ben Johnso	Ipswich To	1348	DF,FW	25	12.6	21-25	-1.03589	Medium	DEF	17.442	4.842
32	Ben White	Arsenal	941	DF	27	58.8	26-30	-1.67215	Low	DEF	51.432	-7.368
33	Bernardo S	Mancheste	2401	MF,FW	30	50	26-30	0.610235	High	MID	40.171	-9.829
34	Bernd Leno	Fulham	3150	GK	33	13.7	30+	1.781128	Very High	GK	11.50307	-2.19693
35	Beto	Everton	1268	FW	27	24.4	26-30	-1.16096	Medium	FWD	31.346	6.946
36	Bilal El Kha	Leicester C	2003	MF	20	40.6	<21	-0.01195	High	MID	44.413	3.813
37	Boubacar	Aston Villa	1463	MF	25	46.3	21-25	-0.85612	Medium	MID	40.652	-5.648
38	Boubakary	Leicester C	1960	MF	26	15.4	26-30	-0.07917	Medium	MID	20.526	5.126
39	Brennan Jo	Tottenham	2083	FW	23	71.3	21-25	0.113113	High	FWD	66.015	-5.285
40	Bruno Ferr	Mancheste	2758	MF	30	54.8	26-30	1.168324	High	MID	40.692	-14.108
41	Bruno Guir	Newcastle	3002	MF	27	83.2	26-30	1.549763	Very High	MID	68.497	-14.703
42	Bryan Mbe	Brentford	3144	FW	25	57.2	21-25	1.771748	Very High	FWD	51.484	-5.716
43	Bukayo Sa	Arsenal	1539	FW,MF	23	101.3	21-25	-0.73731	Medium	FWD	78.699	-22.601
44	Caleb Oko	Leicester C	1124	DF	23	16.7	21-25	-1.38607	Medium	DEF	32.962	16.262
45	Callum Hu	Nott'ham	2152	FW,MF	24	30.2	21-25	0.220979	High	FWD	39.579	9.379
46	Calvin Bas	Fulham	2894	DF	25	29.9	21-25	1.380929	High	DEF	34.007	4.107
47	Cameron A	Southampt	1373	FW	23	18.6	21-25	-0.99681	Medium	FWD	46.226	27.626

5.5. A method, feature and model for estimating player values?

Proposed Method for Estimating Player Transfer Values

We use a Random Forest Regressor to estimate player transfer values. This model is effective for structured data, capable of modeling complex, nonlinear relationships between features and target values. It also reduces overfitting by averaging predictions from multiple decision trees and offers insights into feature importance, supporting transparent and interpretable predictions.

Why Random Forest?

- Handles complex interactions between features.
- Robust against noise and overfitting.
- Provides built-in feature importance.
- Performs well with medium-sized datasets.

Selected Features:

- Minutes_Normalized: reflects actual playing time.
- Age & Age_Group: younger players often hold higher value.
- Experience_Level: represents consistency and maturity.
- Position_Type: positions impact market price (e.g., FWD vs. GK).

Evaluation & Application:

- Evaluated using RMSE and R^2 Score.
- Used to identify:
  - Players with undervalued market prices (buy targets)
  - Players overvalued compared to predicted value (sell candidates)

This method supports data-driven transfer strategies and value optimization.

6. Conclusion

Through the completion of this assignment, I have gained many practical and essential skills in the field of data science—from handling real-world data to applying machine learning models. The knowledge and experience acquired span across multiple key areas:

6.1. Data Collection (Web Scraping)

- I mastered the process of collecting structured data from dynamic websites using Selenium and BeautifulSoup.

- I learned how to handle complex HTML structures (nested tables, multi-level headers) and how to standardize and filter data to meet analysis requirements.

- I developed the mindset to organize data logically, making it easier to process in subsequent steps.

6.2. Descriptive Statistical Analysis

- I learned to use Pandas and NumPy to calculate overall and group-based statistics (e.g., by team).

- I applied methods for comparing top/bottom performers, computing medians, means, and standard deviations.

- I gained experience visualizing data using histogram charts, which helped me detect distribution patterns and trends.

6.3. Player Clustering (Clustering)

- I understood and applied the K-means algorithm to group data based on performance characteristics.

- I used PCA for dimensionality reduction and visualization, which made the clustering results clearer.

- I became familiar with evaluating unsupervised models using the silhouette score, which helped in determining the optimal number of clusters.

6.4. Player Value Prediction (Machine Learning)

- I learned how to collect additional data from external sources and merge it using fuzzy matching—an essential skill when working with data from multiple systems.

- I developed techniques for preprocessing data for ML models: normalization, labeling, and feature engineering.

- I worked with Random Forest, evaluated the model using RMSE and R^2, and understood the significance of feature importance.

- I learned to create recommendation reports based on the difference between predicted and actual values, providing meaningful insights for decision-making.

Final Thoughts

This assignment was a comprehensive learning journey that helped me understand the full process of carrying out a data analysis project—from data collection, processing, analysis, and modeling to visualization and drawing conclusions. It is a valuable experience that I can apply to future real-world projects.

7. Acknowledgements

I would like to extend my sincere gratitude to Mr. Kim Nguyen Bach, the instructor of the Python Programming course, for his dedicated teaching and guidance throughout the course and during the completion of this assignment.

This project has given me the opportunity to experience the full lifecycle of a real-world data science project—from data collection, preprocessing, and statistical analysis to modeling, visualization, and drawing meaningful conclusions. It has been an invaluable experience that helped me improve my programming skills, analytical thinking, and understanding of how artificial intelligence and big data are applied in professional sports—a rapidly growing and highly promising field.

Through this assignment, I also learned how to manage a project from start to finish, work with raw and unstructured data, and overcome common real-world challenges such as missing data, inconsistency, and merging from multiple sources.

Despite my best efforts to complete this assignment thoroughly, I understand that there may still be shortcomings. I sincerely look forward to receiving feedback from my instructor in order to continue improving my skills in the future.

Thank you very much!