BỘ KHOA HỌC VÀ CÔNG NGHỆ

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: LẬP TRÌNH PYTHON

Giảng Viên: Kim Ngọc Bách

Họ và tên:	Nguyễn Trung Tín
Mã sinh viên:	B23DCVT416
Lớp:	D23CQCE04-B

Table of Contents

Problem I

1. Claw Data (LayData.py)

1.1 Objective Overview

- Automate the extraction of player statistics from FBref.com for the 2024 2025 Premier League season, speci%cally targeting various performance categories like standard stats, shooting, passing, goalkeeping, defense, etc. Each category will be saved in a separate CSV %le.

1.2 Technology Description

- Selenium: Automates browser interaction to scrape dynamic content from websites.
- webdriver-manager: Manages the ChromeDriver installation for Selenium.
- BeautifulSoup: Parses HTML content to extract relevant data from web pages.
- CSV Module: Used for writing extracted data into CSV format

```python
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from bs4 import BeautifulSoup, Comment
import time
import csv
```

1.3 Code Breakdown

- write_csv(): Saves extracted data into CSV with or without a header.

```python
# Utility: Save extracted table data to a CSV file
def write_csv(filename, data, header=None):
    with open(filename, mode="w", encoding="utf-8-sig", newline="") as fout:
        writer = csv.writer(fout)
        if header:
            writer.writerow(header)
        writer.writerows(data)
```

- extract_headers(): Extracts table headers from the last row of <thead>.

```python
# Utility: Extract the last row of the table header (<thead>)
def extract_headers(table):
    thead = table.find("thead")
    if not thead:
        return []
    last_row = thead.find_all("tr")[-1]
    return [cell.get_text(strip=True).replace('\xa0', ' ') for cell in last_row.find_all(["th", "td"])]
```

- Web Scraping:
    - Uses Selenium to load pages, waits for content to load.
    - Parses HTML with BeautifulSoup, extracts tables by their IDs.
- Data Processing: Cleans table data, ensuring consistency
- Saving Data: Saves each table's data in a separate CSV file

```python
# Save each scraped table as a separate CSV file
for table_id, content in tables.items():
    filename = f"{table_id}.csv"
    write_csv(filename, content["data"], header=content["header"])
    print(f"💾 Saved {filename} ({len(content['data'])} rows)")
```

1.4 Output

- Generates CSV files for each category (e.g., stats_standard.csv, stats_shooting.csv) containing player statistics.

```
stats_defense.csv      M
stats_gca.csv          M
stats_keeper.csv       M
stats_misc.csv         M
stats_passing.csv      M
stats_possession.csv   M
stats_shooting.csv     M
stats_standard.csv     M
```

2. Merge Data (EPLPlayer.py)

2.1 Objective Overview

- Merge multiple CSV files containing different categories of football player statistics (e.g., shooting, passing, possession, defense) into a unified dataset.

- Ensure data consistency, avoid column name collisions, and filter for players who have played more than 90 minutes in total.
- Export the final cleaned and merged dataset to results.csv for further analysis or modeling.

2.2 Technology Description

- Pandas: Primary library for reading, transforming, merging, and exporting tabular data.
- OS: Used to verify the existence of input files before processing.
- Custom Module (XuLyData.py): Contains utility functions for data cleaning, column renaming, and header normalization.
    - def clean_minutes_column() : Normalize Min column (playing minutes) to numeric type, remove commas and spaces

```python
def clean_minutes_column(df: pd.DataFrame) -> pd.DataFrame:
    if 'Min' in df.columns:
        df['Min'] = (
            df['Min']
            .astype(str)
            .str.replace(",", "")
            .str.strip()
            .pipe(pd.to_numeric, errors='coerce')
        )
    return df
```

    - def clean_duplicate_header(): Remove duplicate header lines in the file caused by crawling from FBref (usually the first line is "Rk").

```python
def clean_duplicate_headers(file_list, keyword='Rk'):

    print("\n🔄 CLEANING DUPLICATE HEADERS")
    for path in file_list:
        try:
            raw = pd.read_csv(path, header=None, encoding='utf-8-sig')
            if raw.empty:
                print(f"   ⚠️ {path} is empty — skipped")
                continue

            mask = ~raw.iloc[:,0].astype(str).str.startswith(keyword)
            mask.iloc[0] = True
            cleaned = raw[mask]

            cleaned.to_csv(path, index=False, header=False, encoding='utf-8-sig')
            print(f"   ✅ {path}: kept {len(cleaned)} rows")
        except Exception as e:
            print(f"   ❌ Error cleaning {path}: {e}")
```

- def sort_and_renumber(): Sort players by first name (Player) and renumber the Rk column if it exists.

```python
def sort_and_renumber(file_list, sort_column='Player'):

    print("\n🔢 SORTING & RENUMBERING")
    for path in file_list:
        try:
            df = pd.read_csv(path, encoding='utf-8-sig')
            if sort_column not in df.columns:
                print(f"   ⚠️ {path} has no '{sort_column}' column — skipped")
                continue

            df = df.sort_values(sort_column)
            if 'Rk' in df.columns:
                df['Rk'] = range(1, len(df)+1)

            df.to_csv(path, index=False, encoding='utf-8-sig')
            print(f"   ✅ Sorted {path}")
        except Exception as e:
            print(f"   ❌ Error sorting {path}: {e}")
```

- def rename_columns(): Many columns in tables have the same name, so rename the columns to be consistent between tables with the same column name but different meanings.

```python
def rename_columns(mappings):

    print("\n📝 RENAMING COLUMNS")
    for inp, out, old, new in mappings:
        try:
            df = pd.read_csv(inp, encoding='utf-8-sig')
            if df.empty:
                print(f"   ⚠️ {inp} is empty — skipped")
                continue

            if old not in df.columns:
                print(f"   ⚠️ '{old}' not found in {inp} — skipped")
                continue

            df = df.rename(columns={old: new})
            df.to_csv(out, index=False, encoding='utf-8-sig')
            print(f"   ✅ {inp}: {old} → {new}")
        except Exception as e:
            print(f"   ❌ Error renaming in {inp}: {e}")
```

- def normalize_pos_column(): Normalize Pos column for later processing.

```python
def normalize_pos_column(df: pd.DataFrame) -> pd.DataFrame:

    if 'Pos' in df.columns:
        df['Pos'] = (
            df['Pos']
            .astype(str)
            .str.replace('"', '')
            .str.replace(',', '-')
        )
    return df
```

2.3 Code Breakdown

- FILE_LIST: Lists all raw CSVs that contain the statistics to be merged.

```python
FILE_LIST = [
    "stats_standard.csv",
    "stats_shooting.csv",
    "stats_possession.csv",
    "stats_passing.csv",
    "stats_misc.csv",
    "stats_keeper.csv",
    "stats_gca.csv",
    "stats_defense.csv",
]
```

- COLUMN_RENAME_MAPPING: Handles potential column name conflicts across different files by mapping old names to new, unique ones.

```python
COLUMN_RENAME_MAPPING = [
    ("stats_misc.csv","stats_misc.csv","Lost","Lostm"),
    ("stats_standard.csv","stats_standard.csv","PrgC","PrgCs"),
    ("stats_standard.csv","stats_standard.csv","PrgP", "PrgPs"),
    ("stats_standard.csv","stats_standard.csv","PrgR", "PrgRs"),
    ("stats_passing.csv", "stats_passing.csv", "PrgP","PrgPp"),
    ("stats_passing.csv", "stats_passing.csv", "1/3", "Pto1/3"),
    ("stats_defense.csv", "stats_defense.csv", "Lost","Lostd"),
    ("stats_defense.csv", "stats_defense.csv", "Att","Attd"),
    ("stats_possession.csv","stats_possession.csv","Att","Attp"),
    ("stats_possession.csv","stats_possession.csv","PrgC","PrgCp"),
    ("stats_possession.csv","stats_possession.csv","1/3","Cto1/3"),
    ("stats_possession.csv","stats_possession.csv","PrgR","PrgRp"),
    ("stats_possession.csv","stats_possession.csv","Lost","Lostm"),
]
```

- HeaderGroups & FileMap: Define which columns to keep from each file and the corresponding file names.

```
# Define headers to retain for each group
HeaderGroups = {
    'standard': ["Player", "Nation", "Squad", "Pos", "Age", "MP", "Starts", "Min",
                 "Gls", "Ast", "CrdY", "CrdR", "xG", "xAG", "PrgCs", "PrgPs", "PrgRs",
                 "Gls.1", "Ast.1", "xG.1", "xAG.1"],
    'shooting': ["Player", "Squad", "SoT%", "SoT/90", "G/Sh", "Dist"],
    'passing': ["Player", "Squad", "Cmp", "Cmp%", "TotDist", "Cmp%.1", "Cmp%.2", "Cmp%.3",
                "KP", "Pto1/3", "PPA", "CrsPA", "PrgPp"],
    'gca': ["Player", "Squad", "SCA", "SCA90", "GCA", "GCA90"],
    'defense': ["Player", "Squad", "Tkl", "TklW", "Attd", "Lostd", "Blocks", "Sh", "Pass", "Int"],
    'possession': ["Player", "Squad", "Touches", "Def Pen", "Def 3rd", "Mid 3rd", "Att 3rd", "Att Pen",
                   "Attp", "Succ%", "Tkld%", "Carries", "PrgDist", "PrgCp", "Cto1/3", "CPA", "Mis", "Dis", "Rec", "PrgRp"],
    'misc': ["Player", "Squad", "Fls", "Fld", "Off", "Crs", "Recov", "Won", "Lostm", "Won%"],
    'goalkeeping': ["Player", "Squad", "GA90", "Save%", "CS%", "Save%.1"]
}

# Map each group name to its corresponding CSV filename
FileMap = {
    'standard': 'stats_standard.csv',
    'shooting': 'stats_shooting.csv',
    'passing': 'stats_passing.csv',
    'gca': 'stats_gca.csv',
    'defense': 'stats_defense.csv',
    'possession': 'stats_possession.csv',
    'misc': 'stats_misc.csv',
    'goalkeeping': 'stats_keeper.csv'
}
```

- Preprocessing:
  - sort_and_renumber(): Sorts files and resets headers if necessary.
  - clean_duplicate_headers(): Removes redundant headers that may exist within data rows.
  - rename_columns(): Renames conflicting column names as defined in the mapping.

```
# Sort files, remove duplicate headers, and rename conflicting columns
XL.sort_and_renumber(FILE_LIST)
XL.clean_duplicate_headers(FILE_LIST)
XL.rename_columns(COLUMN_RENAME_MAPPING)
```

- Base Dataset Creation:
  - Loads stats_standard.csv

```
try:
    # Load base stats from standard file
    std_df = pd.read_csv('stats_standard.csv')

    # Ensure required columns exist
    required_columns = ['PrgCs', 'PrgPs', 'PrgRs']
    for col in required_columns:
        if col not in std_df.columns:
            print(f"⚠ Column {col} not found. Skipping...")
```

- Cleans and normalizes key columns (Min, Pos).

```
# Standardize minutes and position format
std_df = XL.clean_minutes_column(std_df)
std_df = XL.normalize_pos_column(std_df)
```

- Filters out players with total minutes played ≤ 90.

```
# Filter players with total minutes > 90
player_total_min = std_df.groupby('Player')['Min'].sum()
valid_players = player_total_min[player_total_min > 90].index.tolist()
std_df = std_df[(std_df['Player'].isin(valid_players)) & (std_df['Min'] > 90)]
```

- Selects and saves only the required columns de%ned under standard group.

```
    # Select standard columns only
    base_df = std_df[HeaderGroups['standard']].copy()

    print(f"✅ Found {len(base_df)} players with more than 90 minutes.")
except Exception as e:
    print(f"❌ Error processing 'stats_standard.csv': {e}")
    exit()
```

- Merging Remaining Groups:
  - Iterates over all remaining groups (e.g., shooting, passing, defense).
  - Filters players and columns, then merges data with the base using Player and Squad as keys.
  - Skips any %le not found or failing to process.

```python
# Merge all remaining data groups into base_df
for group, headers in HeaderGroups.items():
    if group == 'standard':
        continue  # Already handled

    file = FileMap[group]
    print(f"\n📄 Processing file: {file}")

    if not os.path.exists(file):
        print(f"⚠️ File not found: {file}. Skipping...")
        continue

    try:
        df = pd.read_csv(file)
        df = XL.clean_minutes_column(df)
        df = XL.normalize_pos_column(df)

        # Filter rows by minimum minutes and valid players
        if 'Min' in df.columns:
            df = df[df['Min'] > 90]
        df = df[df['Player'].isin(base_df['Player'])]

        # Keep relevant headers only
        df = df[headers].copy()

        # Merge with base dataframe on Player and Squad
        base_df = pd.merge(base_df, df, on=["Player", "Squad"], how="left")

    except Exception as e:
        print(f"❌ Error merging {file}: {e}")
```

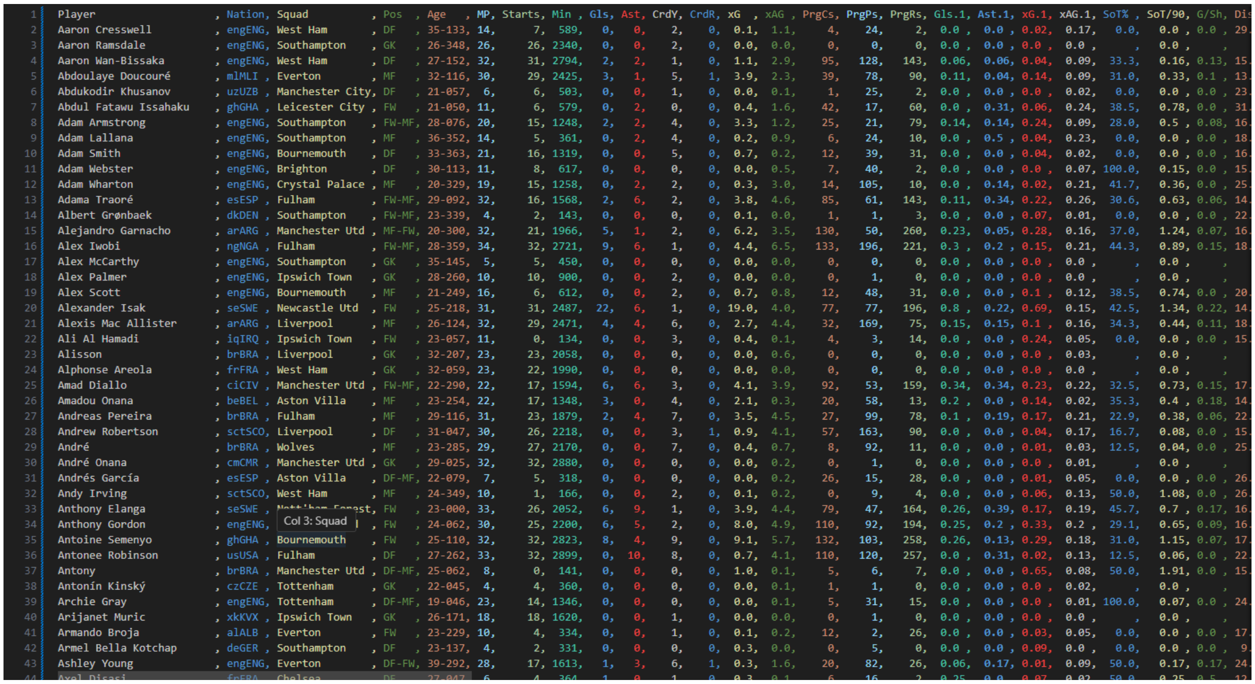
- Post-Processing and Export:
    - Fills missing values in object-type columns with "Na".
    - Writes the final cleaned dataset to results.csv.

```python
try:
    # Replace missing object-type values with 'Na'
    for col in base_df.select_dtypes(include=['object']).columns:
        base_df[col] = base_df[col].fillna("Na")

    # Export the final merged result to CSV
    base_df.to_csv("results.csv", index=False, encoding='utf-8-sig')

    print(f"\n✅ Final dataset saved: results.csv")
    print(f"👥 Total unique players: {base_df['Player'].nunique()}")
    print(f"📊 Total rows: {len(base_df)}")
    print(f"📌 Sample players: {base_df['Player'].drop_duplicates().head(5).tolist()} ...")

except Exception as e:
    print(f"❌ Error saving results: {e}")
```

2.4 Output

- A unified results.csv file containing cleaned, de-duplicated, and merged player statistics across all performance categories.

- Ensures only players with signi%cant play time (over 90 minutes) are included.
- Data is ready for use in downstream tasks such as modeling, analysis, or visualization.



```
Player                 , Nation, Squad           , Pos  , Age   , MP, Starts, Min , Gls, Ast, CrdY, CrdR, xG  , xAG , PrgCs, PrgPs, PrgRs, Gls.1, Ast.1, xG.1, xAG.1, SoT% , SoT/90, G/Sh, Dis
Aaron Cresswell        , engENG, West Ham        , DF   , 35-133, 14,  7,  589,  0,  0, 2, 0,  0.1, 1.1,   4,  24,   2, 0.0 , 0.0 , 0.02, 0.17,   0.0, 0.0 , 0.0 , 29
Aaron Ramsdale         , engENG, Southampton     , GK   , 26-348, 26, 26, 2340,  0,  0, 2, 0,  0.0, 0.0,   0,   0,   0, 0.0 , 0.0 , 0.0 , 0.0 ,      , 0.0 ,     ,
Aaron Wan-Bissaka      , engENG, West Ham        , DF   , 27-152, 32, 31, 2794,  2,  2, 1, 0,  1.1, 2.9,  95, 128, 143, 0.06, 0.06, 0.04, 0.09,  33.3, 0.16, 0.13, 15
Abdoulaye Doucouré     , mlMLI , Everton         , MF   , 32-116, 30, 29, 2425,  3,  1, 5, 1,  3.9, 2.3,  39,  78,  90, 0.11, 0.04, 0.14, 0.09,  31.0, 0.33, 0.1 , 13
Abdukodir Khusanov     , uzUZB , Manchester City , DF   , 21-057,  6,  6,  503,  0,  0, 1, 0,  0.0, 0.1,   1,  25,   2, 0.0 , 0.0 , 0.0 , 0.02,   0.0, 0.0 , 0.0 , 23
Abdul Fatawu Issahaku  , ghGHA , Leicester City  , FW   , 21-050, 11,  6,  579,  0,  2, 0, 0,  0.4, 1.6,  42,  17,  60, 0.0 , 0.31, 0.06, 0.24,  38.5, 0.78, 0.0 , 31
Adam Armstrong         , engENG, Southampton     , FW-MF, 28-076, 20, 15, 1248,  2,  2, 4, 0,  3.3, 1.2,  25,  21,  79, 0.14, 0.14, 0.24, 0.09,  28.0, 0.5 , 0.08, 16
Adam Lallana           , engENG, Southampton     , MF   , 36-352, 14,  5,  361,  0,  2, 4, 0,  0.2, 0.9,   6,  24,  10, 0.0 , 0.5 , 0.04, 0.23,   0.0, 0.0 , 0.0 , 18
Adam Smith             , engENG, Bournemouth     , DF   , 33-363, 21, 16, 1319,  0,  0, 5, 0,  0.7, 0.2,  12,  39,  31, 0.0 , 0.0 , 0.04, 0.02,   0.0, 0.0 , 0.0 , 16
Adam Webster           , engENG, Brighton        , DF   , 30-113, 11,  8,  617,  0,  0, 0, 0,  0.0, 0.5,   7,  40,   2, 0.0 , 0.0 , 0.0 , 0.07, 100.0, 0.15, 0.0 , 15
Adam Wharton           , engENG, Crystal Palace  , MF   , 20-329, 19, 15, 1258,  0,  2, 2, 0,  0.3, 3.0,  14, 105,  10, 0.0 , 0.14, 0.02, 0.21,  41.7, 0.36, 0.0 , 25
Adama Traoré           , esESP , Fulham          , FW-MF, 29-092, 32, 16, 1568,  2,  6, 2, 0,  3.8, 4.6,  85,  61, 143, 0.11, 0.34, 0.22, 0.26,  30.6, 0.63, 0.06, 14
Albert Grønbaek        , dkDEN , Southampton     , FW-MF, 23-339,  4,  2,  143,  0,  0, 0, 0,  0.1, 0.0,   1,   1,   3, 0.0 , 0.0 , 0.07, 0.01,   0.0, 0.0 , 0.0 , 22
Alejandro Garnacho     , arARG , Manchester Utd  , MF-FW, 20-300, 32, 21, 1966,  5,  1, 2, 0,  6.2, 3.5, 130,  50, 260, 0.23, 0.05, 0.28, 0.16,  37.0, 1.24, 0.07, 16
Alex Iwobi             , ngNGA , Fulham          , FW-MF, 28-359, 34, 32, 2721,  9,  6, 1, 0,  4.4, 6.5, 133, 196, 221, 0.3 , 0.2 , 0.15, 0.21,  44.3, 0.89, 0.15, 18
Alex McCarthy          , engENG, Southampton     , GK   , 35-145,  5,  5,  450,  0,  0, 0, 0,  0.0, 0.0,   0,   0,   0, 0.0 , 0.0 , 0.0 , 0.0 ,      , 0.0 ,     ,
Alex Palmer            , engENG, Ipswich Town    , GK   , 28-260, 10, 10,  900,  0,  0, 2, 0,  0.0, 0.0,   0,   1,   0, 0.0 , 0.0 , 0.0 , 0.0 ,      , 0.0 ,     ,
Alex Scott             , engENG, Bournemouth     , MF   , 21-249, 16,  6,  612,  0,  0, 2, 0,  0.7, 0.8,  12,  48,  31, 0.0 , 0.0 , 0.1 , 0.12,  38.5, 0.74, 0.0 , 20
Alexander Isak         , seSWE , Newcastle Utd   , FW   , 25-218, 31, 31, 2487, 22,  6, 1, 0, 19.0, 4.0,  77,  77, 196, 0.8 , 0.22, 0.69, 0.15,  42.5, 1.34, 0.22, 14
Alexis Mac Allister    , arARG , Liverpool       , MF   , 26-124, 32, 29, 2471,  4,  4, 6, 0,  2.7, 4.4,  32, 169,  75, 0.15, 0.15, 0.1 , 0.16,  34.3, 0.44, 0.11, 18
Ali Al Hamadi          , iqIRQ , Ipswich Town    , FW   , 23-057, 11,  0,  134,  0,  0, 3, 0,  0.4, 0.1,   4,   3,  14, 0.0 , 0.0 , 0.24, 0.05,   0.0, 0.0 , 0.0 , 15
Alisson                , brBRA , Liverpool       , GK   , 32-207, 23, 23, 2058,  0,  0, 0, 0,  0.0, 0.6,   0,   0,   0, 0.0 , 0.0 , 0.0 , 0.03,      , 0.0 ,     ,
Alphonse Areola        , frFRA , West Ham        , GK   , 32-059, 23, 22, 1990,  0,  0, 0, 0,  0.0, 0.0,   0,   0,   0, 0.0 , 0.0 , 0.0 , 0.0 ,      , 0.0 ,     ,
Amad Diallo            , ciCIV , Manchester Utd  , FW-MF, 22-290, 22, 17, 1594,  6,  6, 3, 0,  4.1, 3.9,  92,  53, 159, 0.34, 0.34, 0.23, 0.22,  32.5, 0.73, 0.15, 17
Amadou Onana           , beBEL , Aston Villa     , MF   , 23-254, 22, 17, 1348,  3,  0, 4, 0,  2.1, 0.3,  20,  58,  13, 0.2 , 0.0 , 0.14, 0.02,  35.3, 0.4 , 0.18, 14
Andreas Pereira        , brBRA , Fulham          , MF   , 29-116, 31, 23, 1879,  2,  4, 7, 0,  3.5, 4.5,  27,  99,  78, 0.1 , 0.19, 0.17, 0.21,  22.9, 0.38, 0.06, 22
Andrew Robertson       , sctSCO, Liverpool       , DF   , 31-047, 30, 26, 2218,  0,  0, 3, 1,  0.9, 4.1,  57, 163,  90, 0.0 , 0.0 , 0.04, 0.17,  16.7, 0.08, 0.0 , 15
André                  , brBRA , Wolves          , MF   , 23-285, 29, 27, 2170,  0,  0, 7, 0,  0.4, 0.7,   8,  92,  11, 0.0 , 0.0 , 0.01, 0.03,  12.5, 0.04, 0.0 , 25
André Onana            , cmCMR , Manchester Utd  , GK   , 29-025, 32, 32, 2880,  0,  0, 0, 0,  0.0, 0.2,   0,   1,   0, 0.0 , 0.0 , 0.0 , 0.01,      , 0.0 ,     ,
Andrés García          , esESP , Aston Villa     , DF-MF, 22-079,  7,  5,  318,  0,  0, 0, 0,  0.0, 0.2,  26,  15,  28, 0.0 , 0.0 , 0.01, 0.05,   0.0, 0.0 , 0.0 , 26
Andy Irving            , sctSCO, West Ham        , MF   , 24-349, 10,  1,  166,  0,  0, 2, 0,  0.1, 0.2,   0,   9,   4, 0.0 , 0.0 , 0.06, 0.13,  50.0, 1.08, 0.0 , 26
Anthony Elanga         , seSWE , Nott'ham Forest , FW   , 23-000, 33, 26, 2052,  6,  9, 1, 0,  3.9, 4.4,  79,  47, 164, 0.26, 0.39, 0.17, 0.19,  45.7, 0.7 , 0.17, 16
Anthony Gordon         , engENG, Col 3: Squad    , FW   , 24-062, 30, 25, 2200,  6,  5, 2, 0,  8.0, 4.9, 110,  92, 194, 0.25, 0.2 , 0.33, 0.2 ,  29.1, 0.65, 0.09, 16
Antoine Semenyo        , ghGHA , Bournemouth     , FW   , 25-110, 32, 32, 2823,  8,  4, 9, 0,  9.1, 5.7, 132, 103, 258, 0.26, 0.13, 0.29, 0.18,  31.0, 1.15, 0.07, 17
Antonee Robinson       , usUSA , Fulham          , DF   , 27-262, 33, 32, 2899,  0, 10, 8, 0,  0.7, 4.1, 110, 120, 257, 0.0 , 0.31, 0.02, 0.13,  12.5, 0.06, 0.0 , 22
Antony                 , brBRA , Manchester Utd  , DF-MF, 25-062,  8,  0,  141,  0,  0, 0, 0,  1.0, 0.1,   5,   6,   7, 0.0 , 0.0 , 0.65, 0.08,  50.0, 1.91, 0.0 , 15
Antonín Kinský         , czCZE , Tottenham       , GK   , 22-045,  4,  4,  360,  0,  0, 0, 0,  0.0, 0.1,   1,   1,   0, 0.0 , 0.0 , 0.0 , 0.02,      , 0.0 ,     ,
Archie Gray            , engENG, Tottenham       , DF-MF, 19-046, 23, 14, 1346,  0,  0, 0, 0,  0.0, 0.1,   5,  31,  15, 0.0 , 0.0 , 0.0 , 0.01, 100.0, 0.07, 0.0 , 24
Arijanet Muric         , xkKVX , Ipswich Town    , GK   , 26-171, 18, 18, 1620,  0,  0, 1, 0,  0.0, 0.0,   0,   1,   0, 0.0 , 0.0 , 0.0 , 0.0 ,      , 0.0 ,     ,
Armando Broja          , alALB , Everton         , FW   , 23-229, 10,  4,  334,  0,  0, 1, 0,  0.1, 0.2,  12,   2,  26, 0.0 , 0.0 , 0.03, 0.05,   0.0, 0.0 , 0.0 , 17
Armel Bella Kotchap    , deGER , Southampton     , DF   , 23-137,  4,  2,  331,  0,  0, 0, 0,  0.3, 0.0,   0,   5,   0, 0.0 , 0.0 , 0.09, 0.0 ,   0.0, 0.0 , 0.0 ,  9
Ashley Young           , engENG, Everton         , DF-FW, 39-292, 28, 17, 1613,  1,  3, 6, 1,  0.3, 1.6,  20,  82,  26, 0.06, 0.17, 0.01, 0.09,  50.0, 0.17, 0.17, 24
Axel Disasi            , frFRA , Chelsea         , DF   , 27-047,  6,  4,  364,  1,  0, 1, 0,  0.3, 0.1,   6,  16,   2, 0.25, 0.0 , 0.07, 0.02,  50.0, 0.25, 0.5 , 12
```

Problem II

1. Top 3 players with the highest and lowest scores (Top3Player.py)

1.1 Objective Overview

- Analyze key football player performance metrics from the merged dataset (results.csv).
- Identify the Top-3 highest and lowest players for each metric and save a summary report to a human-readable text file (top3.txt).

1.2 Technology Description

- Pandas: Handles CSV reading, numeric conversion, and ranking operations efficiently.

```
import pandas as pd
```

- Python Built-in File I/O: Used to write the Top-3 results to a text file in structured format.

1.3 Code Breakdown

- METRICS: A curated list of numeric or string-based performance metrics to be evaluated.
- INPUT / OUTPUT: File paths for the input data (results.csv) and output report (top3.txt).
- Loading and Validation:
    - Reads results.csv using UTF-8-SIG encoding.

```
try:
    # Load data from CSV file
    print(f"📁 Loading data from '{INPUT}' ...")
    df = pd.read_csv(INPUT, na_values=["N/a", "NA", "NaN", "-", ""], encoding="utf-8-sig", engine="python")
    if df.empty:
        raise ValueError(f"❌ Input file '{INPUT}' contains no data.")
    print("✅ Data loaded successfully.")
```

    - Validates that the file contains data and that required columns (e.g., Player) exist.

```
# Ensure 'Player' column exists
if "Player" not in df.columns:
    raise KeyError("❌ Input data must contain a 'Player' column.")
```

- Displays error messages for missing input files or broken schemas

- Sanitization and Preprocessing:
  - Fills missing values for each metric.
  - Converts numeric metrics to float, and fills invalid/missing entries with 0.
  - Treats Age as a string for consistent alphaetical sorting.

```python
# Sanitize the columns to handle missing values and convert to numeric
print("🧹 Sanitizing columns ...")
for col in METRICS:
    if col not in df.columns:
        print(f"⚠️ Column '{col}' not found, skipping.")
        continue

    if col == "Age":
        df["Age"] = df["Age"].fillna("0").astype(str)
    else:
        df[col] = pd.to_numeric(df[col], errors="coerce").fillna(0.0)
print("✅ Columns sanitized.")
```

- Top-3 Computation:
  - Filters valid rows
  - Computes Top-3 highest and lowest values using nlargest() and nsmallest() (or string sort for Age)
  - Formats results and adds them to a text buffer
- Output Writing:
  - Writes all metric summaries into a clean, structured file (top3.txt).

```python
    # Write the results to an output file
    with open(OUTPUT, "w", encoding="utf-8") as f:
        f.write("\n".join(lines))

    print(f"✅ Saved Top‑3 report to '{OUTPUT}'")
    print("🎉 Process complete.")

except FileNotFoundError:
    print(f"❌ File not found: {INPUT}")
except Exception as e:
    print(f"❌ Error: {e}")
```

- Each section includes the metric name, Top-3 highest and lowest performers, and separators for readability.

```
# Compute Top-3 highest and lowest for each metric
print("📊 Computing Top-3 for each metric ...")
lines = ["Top-3 HIGHEST & LOWEST for each metric:\n"]

for metric in METRICS:
    if metric not in df.columns:
        continue

    lines.append(f"--- Metric: {metric} ---")
    valid = df[metric].notna()
    if valid.sum() < 3:
        lines.append(f"⚠️ Not enough valid data for '{metric}'.\n")
        continue

    sub = df.loc[valid, ["Player", metric]]

    if metric == "Age":
        high = sub.sort_values(by=metric, ascending=False, key=lambda s: s.str.lower()).head(3)
        low = sub.sort_values(by=metric, ascending=True, key=lambda s: s.str.lower()).head(3)
    else:
        high = sub.nlargest(3, metric)
        low = sub.nsmallest(3, metric)
```

1.4 Output

- A detailed text file top3.txt listing the Top-3 best and worst players per metric in the dataset.
- Provides quick insights for scouting, performance analysis, or reporting.

```
--------------------------------------------------


--- Metric: Gls ---
Top-3 Highest:
           Player  Gls
 Mohamed Salah    27
Erling Haaland    21
Alexander Isak    20

Top-3 Lowest:
               Player  Gls
   Aaron Cresswell    0
    Aaron Ramsdale    0
Abdukodir Khusanov    0
```

2. Median of data (TrungVi.py)

2.1 Objective Overview

- Find the median for each statistic. Calculate the mean and standard deviation for each statistic across all players and for each team

2.2 Technology Description

- pandas: For reading, manipulating, and analyzing tabular data.
- numpy: For numerical operations (used indirectly via pandas).
- csv, sys: To handle system-specific limitations for CSV processing, especially large field sizes.

```python
import pandas as pd
import numpy as np
import csv
import sys
```

2.3 Code Breakdown

- Sets the CSV field size limit using sys.maxsize to accommodate large datasets.

```python
# Ensure CSV field size limit can handle large fields
max_int = sys.maxsize
while True:
    try:
        csv.field_size_limit(max_int)
        break
    except OverflowError:
        max_int = int(max_int / 10)
```

- Reads the CSV file using pandas.read_csv with UTF-8 encoding.

```python
# Reading the data from 'results.csv'
print("📂 Reading data from 'results.csv'...")
try:
    df = pd.read_csv('results.csv', engine='python', encoding='utf-8-sig', skip_blank_lines=True)
```

- Cleans column names by removing newline and carriage return characters.

```python
    # Clean up column names by removing unwanted characters
    df.columns = df.columns.str.strip().str.replace('\n', '').str.replace('\r', '')
    print(f"✅ Successfully read file with {len(df)} records and {len(df.columns)} columns.")
except Exception as e:
    print(f"❌ Error reading file: {e}")
    exit()
```

- Validates whether all necessary performance metrics (like Gls, xAG, Touches, etc.) exist in the dataset.

```
# List of required columns to analyze
required_headers = [
    "Age", "MP", "Starts", "Min", "Gls", "Ast", "CrdY", "CrdR", "xG", "xAG",
    "PrgCs", "PrgPs", "PrgRs", "Gls.1", "Ast.1", "xG.1", "xAG.1", "SoT%", "SoT/90",
    "G/Sh", "Dist", "Cmp", "Cmp%", "TotDist", "Cmp%.1", "Cmp%.2", "Cmp%.3",
    "KP", "Pto1/3", "PPA", "CrsPA", "PrgPp", "SCA", "SCA90", "GCA", "GCA90",
    "Tkl", "TklW", "Attd", "Lostd", "Blocks", "Sh", "Pass", "Int", "Touches",
    "Def Pen", "Def 3rd", "Mid 3rd", "Att 3rd", "Att Pen", "Attp", "Succ%",
    "Tkld%", "Carries", "PrgDist", "PrgCp", "Cto1/3", "CPA", "Mis", "Dis",
    "Rec", "PrgRp", "Fls", "Fld", "Off", "Crs", "Recov", "Won", "Lostm",
    "Won%", "GA90", "Save%", "CS%", "Save%.1"
]

# Check if all required columns are present in the data
missing_headers = [col for col in required_headers if col not in df.columns]
if missing_headers:
    print(f"⚠️ Missing {len(missing_headers)} required headers.")
else:
    print("✅ All required headers are present.")
```

- Drops irrelevant ﬁelds such as Player, Nation, and Pos.

```
# Drop irrelevant columns such as 'Player', 'Nation', 'Pos'
cols_to_drop = ['Player', 'Nation', 'Pos']
df.drop(columns=[col for col in cols_to_drop if col in df.columns], inplace=True, errors='ignore')
```

- Normalizes the Age column by converting values like '22-150' to a decimal form (22 + 150/365).

```
# Normalize 'Age' column: converting age ranges (e.g., '22-150') into numeric values
if 'Age' in df.columns:
    print("🔄 Normalizing 'Age' column...")

    def convert_age(x):
        if pd.isna(x):
            return 0.0
        if isinstance(x, str) and '-' in x:
            try:
                years, days = map(int, x.split('-'))
                return years + days / 365  # Convert range to average age
            except:
                return 0.0
        try:
            return float(x)
        except:
            return 0.0

    df['Age'] = df['Age'].apply(convert_age)
```

- Ensures all numeric columns are properly converted.

```
# Identify numeric columns in the dataset
numeric_cols = [col for col in required_headers if col in df.columns]
print(f"📊 Preparing to compute statistics for {len(numeric_cols)} columns.")
```

- Replaces non-numeric and missing values ('N/A', empty cells, etc.) with 0.0.

```
# Clean data by converting non-numeric or 'N/A' values to 0.0
print("🧹 Cleaning data (N/A → 0.0)...")
df[numeric_cols] = df[numeric_cols].apply(lambda x: pd.to_numeric(x, errors='coerce')).fillna(0.0)
```

- For each performance metric: median, mean, standard deviation
- Once across all players.

```
# DataFrame to store calculated statistics
result_df = pd.DataFrame()

# Function to compute median, mean, and standard deviation for each metric
def calculate_stats(data, team_name):
    stats = {'Team': team_name}
    for col in numeric_cols:
        stats[f'Median of {col}'] = data[col].median()
        stats[f'Mean of {col}'] = data[col].mean()
        stats[f'Std of {col}'] = data[col].std()
    return stats
```

- Separately for each team (using the Squad column).

```
# Calculate global statistics across all players
print("🔍 Computing overall statistics...")
result_df = pd.concat([result_df, pd.DataFrame([calculate_stats(df, 'all')])], ignore_index=True)

# Calculate statistics for each team
if 'Squad' in df.columns:
    print("🏟 Computing stats per team...")
    for team in df['Squad'].unique():
        team_df = df[df['Squad'] == team]
        team_stats = calculate_stats(team_df, team)
        result_df = pd.concat([result_df, pd.DataFrame([team_stats])], ignore_index=True)
```

- Results are written to results2.csv.

```
# Save the results to a new CSV file
output_file = 'results2.csv'
print(f"💾 Saving results to '{output_file}'...")
try:
    result_df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"✅ Saved successfully with {len(result_df)} rows and {len(result_df.columns)} columns.")
except Exception as e:
    print(f"❌ Error saving file: {e}")

print("✅ Finished!")
```

-

- The final file includes:
    - One row for overall statistics (Team = all)
    - One row for each team

2.4 Output

- Output File: results2.csv
- Columns:
    - Team
    - For each metric (e.g., xG, SCA, GA90), three columns: Median of <metric>, Mean of <metric>, Std of <metric>
- Rows:
    - First row: Global stats for all players (Team = all)
    - Following rows: Stats per team (Team = <team name>)

BTLPython > Bai2 > results2.csv > data

Team	Median of Age	Mean of Age	Std of Age	Median of MP	Mean of MP	Std of MP	Median of Starts	Mean of Starts	Std of Starts	Med
all	26.508219178082193	26.875670780029438	4.205357628735091	20.0	19.051652892561982	8.836414011532096	13.5	14.026859504132231	9.782801787872415	
West Ham	28.268493150684932	28.548273972602743	5.01532014101243	20.0	19.2	7.762087348130012	12.0	13.64	9.66902270139025	
Southampton	26.90684931506849	26.90552668871044	4.225866786297974	19.0	16.620689655172413	9.18233472140722	11.0	11.724137931034482	8.564447074655858	
Everton	26.464383561643835	27.904981320049817	5.031962455952982	22.5	19.727272727272727	8.718790972222928	14.5	15.454545454545455	9.703839557850396	
Manchester City	26.61917808219178	26.927232876712328	4.913737779471896	20.0	17.56	8.426545357776618	14.0	13.6	8.190441583537076	
Leicester City	26.68082191780822	27.148050579557427	4.46222726444891	19.0	18.0	8.993330862366847	12.5	13.115384615384615	9.218793513587006	
Bournemouth	25.931506849315067	26.40202501488981	3.83988654120663	22.0	19.782608695652176	8.893084758492924	14.0	14.782608695652174	10.479228466304752	
Brighton	24.383561643835616	25.793099949264334	5.182941078703074	19.0	17.814814814814813	8.831922157879847	9.0	12.62962962962963	9.043264166252515	
Crystal Palace	27.12054794520548	27.105805609915198	3.134913052545247	25.0	20.571428571428573	8.806653328997507	15.0	15.619047619047619	10.99307141101244	
Fulham	28.75890410958904	28.752428393524283	3.351455998959689	23.0	21.681818181818183	8.571194080486569	16.0	15.454545454545455	10.391055094360052	
Manchester Utd	25.660273972602738	26.000876712328772	4.893185885017612	20.0	18.4	9.928914005737655	15.0	13.64	10.016153619695203	
Ipswich Town	26.632876712328766	26.833150684931507	3.155121097020423	17.0	16.1	8.21835624138701	9.0	11.366666666666667	8.999936142813052	
Newcastle Utd	27.420547945205477	28.17409713574097	4.633248816483576	25.5	20.227272727272727	8.8690570615017	15.0	15.0	10.614455552060438	
Liverpool	26.378082191780823	27.143509458577952	3.689684918684410	25.0	22.238095238095237	8.15416925201312	17.0	16.238095238095237	11.04492988617294	
Aston Villa	27.565753424657533	27.071037181996086	4.035940846227701	19.0	17.142857142857142	9.276470775726622	8.5	12.178571428571429	10.505919218437	
Wolves	26.786301369863015	27.607623585467536	3.993813769733957	24.0	20.08695652173913	8.022202392441644	15.0	14.782608695652174	9.472239447886796	
Nott'ham Forest	26.783561643835615	27.089367253750815	3.640179180620415	27.0	22.714285714285715	8.911469335316468	17.0	16.238095238095237	11.592690636365493	
Tottenham	25.572602739726026	25.591577879249115	4.538458558710817	18.0	17.185185185185187	8.871029066300974	13.0	12.592592592592593	7.776984086487608	
Chelsea	24.26027397260274	24.33413698630137	2.195618284439809	18.0	17.96	9.680220383166215	12.0	13.64	10.094883192324055	
Brentford	25.221917808219178	26.40013698630137	3.81103042875020	24.0	21.7	9.744634151000339	20.0	17.05	11.5506892024948	
Arsenal	26.80958904109589	26.42963885429639	3.822743904750704	21.0	20.545454545454547	7.657721112752321	15.0	15.5	9.752899446782358	

3. A histogram plots the distribution of each statistic for all players in the league and each team.

3.1 For each team (VeTeam.py)

3.1.1 Objective Overview

- Normalize the data and plot a histogram showing the distribution of each team.
- Since creating plots for each team requires precise statistics, I have performed data calculations to ensure a more objective and accurate visualization process (ChiSoTheoTeam.py).

- Reading Data: The code reads player data from a CSV file (results.csv).

```python
# Load player data from CSV
df = pd.read_csv('results.csv')
```

- Identifying Numeric Columns: The code checks each column to identify which ones contain numeric data (ignoring non-statistic columns like 'Player', 'Nation', 'Squad', and 'Pos').

```python
# Define non-statistic columns
non_stat_cols = ['Player', 'Nation', 'Squad', 'Pos']

# Identify numeric columns by attempting conversion
numeric_cols = []
for col in df.columns:
    if col in non_stat_cols:
        continue
    try:
        pd.to_numeric(df[col].replace('N/a', 0.0), errors='raise')
        numeric_cols.append(col)
    except Exception:
        continue
```

- Cleaning Data: It replaces 'N/a' with 0.0 and converts columns to the float type.

```python
# Clean and convert numeric columns
df[numeric_cols] = df[numeric_cols].replace('N/a', 0.0).astype(float)
```

- Calculating Averages: It groups the data by 'Squad' and calculates the average for each numeric column.

```python
# Calculate average stats per team
squad_avg = df.groupby('Squad')[numeric_cols].mean(numeric_only=True)
```

- Saving Data: The calculated averages are saved to a new CSV file (ChiSoTeam.csv)

```python
# Save to output CSV
output_path = 'ChiSoTeam.csv'
squad_avg.to_csv(output_path, encoding='utf-8-sig')

print(f"✅ Saved average team stats to {output_path}")
```

- Output:

Squad	MP	Starts	Min	Gls	Ast	CrdY	CrdR	xG	xAG
Arsenal	20.545454545454547	15.5	1384.7727272727273	2.4545454545454546	1.9545454545454546	2.8181818181818183	0.22727272727272727	2.2863636363636366	1.727272727
Aston Villa	17.142857142857142	12.178571428571429	1093.9285714285713	1.6071428571428572	1.2857142857142858	2.25	0.07142857142857142	1.6642857142857144	1.25
Bournemouth	19.782608695652176	14.782608695652174	1323.6521739130435	2.1739130434782608	1.608695652173913	3.3043478260869565	0.043478260869565216	2.526086956521739	1.678260869
Brentford	21.7	17.05	1525.0	2.55	1.6	2.1	0.05	2.415	1.735
Brighton	17.814814814814813	12.62962962962963	1131.8148148148148	1.7407407407407407	1.1481481481481481	2.4074074074074074	0.07407407407407407	1.7037037037037037	1.188888888
Chelsea	17.96	13.64	1220.72	2.12	1.6	3.52	0.04	2.376	1.796
Crystal Palace	20.571428571428573	15.619047619047619	1401.2857142857142	1.6666666666666667	1.380952380952381	2.9047619047619047	0.14285714285714285	2.1619047619047618	1.657142857
Everton	19.727272727272727	15.454545454545455	1386.5	1.3636363636363635	0.8636363636363636	2.909090909090909	0.09090909090909091	1.55	1.177272727
Fulham	21.681818181818183	15.454545454545455	1388.8181818181818	2.090909090909091	1.7727272727272727	3.0	0.09090909090909091	1.9	1.459090909
Ipswich Town	16.1	11.366666666666667	1018.5333333333333	1.0	0.7333333333333333	2.6333333333333333	0.1	0.9933333333333334	0.663333333
Leicester City	18.0	13.115384615384615	1178.576923076923	0.9615384615384616	0.7692307692307693	2.6923076923076925	0.0	1.0115384615384615	0.807692307
Liverpool	22.238095238095237	16.238095238095237	1454.952380952381	3.4285714285714284	2.5714285714285716	2.761904761904762	0.09523809523809523	3.314285714285714	2.376190476
Manchester City	17.56	13.6	1223.32	2.24	1.64	2.12	0.04	2.2	1.783999999
Manchester Utd	18.4	13.64	1216.4	1.4	0.92	2.76	0.12	1.6840000000000002	1.24
Newcastle Utd	20.227272727272727	15.0	1342.5	2.3181818181818183	1.7272727272727273	2.409090909090909	0.045454545454545456	2.3272727272727276	1.672727272
Nott'ham Forest	22.714285714285715	16.238095238095237	1456.7619047619048	2.380952380952381	1.7142857142857142	3.1904761904761907	0.09523809523809523	1.880952380952381	1.361904761
Southampton	16.620689655172413	11.724137931034482	1049.0	0.7586206896551724	0.4827586206896552	2.586206896551724	0.10344827586206896	0.9965517241379311	0.744827586
Tottenham	17.185185185185187	12.592592592592593	1131.111111111111	2.037037037037037	1.5555555555555556	2.2222222222222223	0.037037037037037035	1.9074074074074074	1.437037037
West Ham	19.2	13.64	1223.12	1.32	0.8	2.68	0.12	1.5919999999999999	1.092
Wolves	20.08695652173913	14.782608695652174	1327.0	1.8695652173913044	1.5217391304347827	3.0	0.08695652173913043	1.5521739130434784	1.260869565

3.1.2 Technology Description

- pandas: For reading, cleaning, and manipulating tabular data e ciently
- matplotlib.pyplot: For creating histograms to visualize distributions of player statistics.
- csv, sys: Handle large CSV ﬁeld sizes and system-level constraints.

3.1.3 Code Breakdown

- The script reads player statistics from results.csv.

```
# Load player statistics from the CSV file
df = pd.read_csv("results.csv")

# Columns to exclude from numerical processing
skip_headers = ['Player', 'Nation', 'Squad', 'Pos']
```

- Non-numeric placeholders like "N/a" are replaced with 0.00.

```
# Replace invalid values labeled as "N/a" with 0.00
df.replace("N/a", 0.00, inplace=True)
```

- The "Age" column, originally in the "years-days" format, is normalized into a decimal value (e.g., 20-200 → 20.55).

```
# Function to normalize age from the "years-days" format to float (e.g., "20-200" → 20.55)
def normalize_age(age_str):
    try:
        if isinstance(age_str, str) and '-' in age_str:
            y, d = map(int, age_str.split('-'))
            return round(y + d / 365, 2)
        return float(age_str)
    except:
        return 0.0

# Normalize the 'Age' column if it exists in the dataset
if 'Age' in df.columns:
    df['Age'] = df['Age'].apply(normalize_age)
```

- All columns (except identiﬁers like "Player", "Nation", "Squad", "Pos") are converted to ﬂoats.
- Only numeric columns are used for histogram generation.

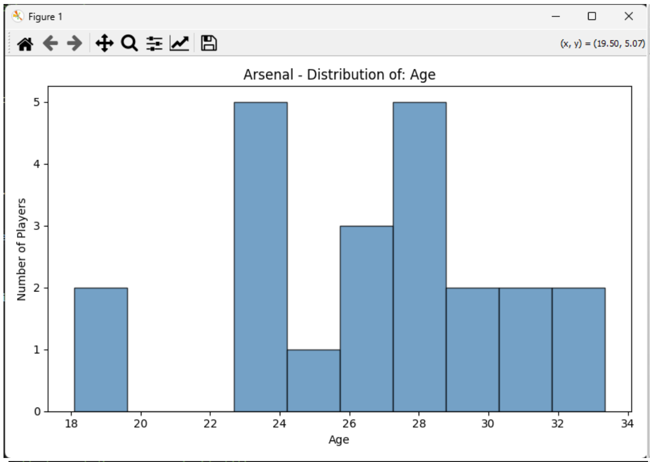
```
# Convert all numeric columns to float, excluding non-numeric identifiers
for col in df.columns:
    if col not in skip_headers:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0.0)

# Identify numeric columns to be used in plotting
numeric_columns = [col for col in df.columns if col not in skip_headers]

# Get the list of all unique teams from the 'Squad' column
teams = sorted(df['Squad'].dropna().unique())
```

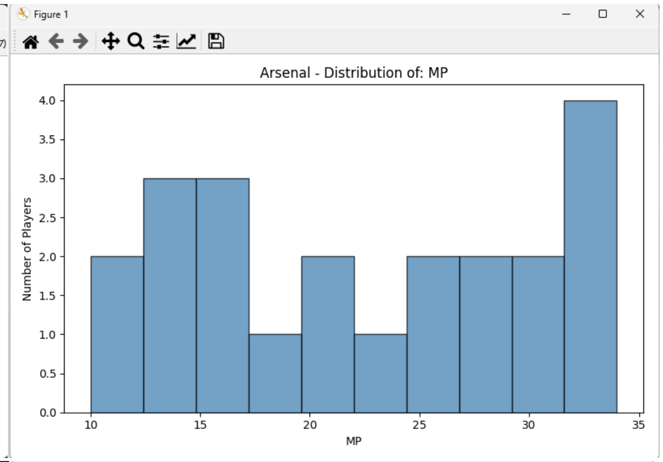
- For each team in the "Squad" column, a histogram is plotted for every numeric metric that has more than one unique value to ensure meaningful visualizations.

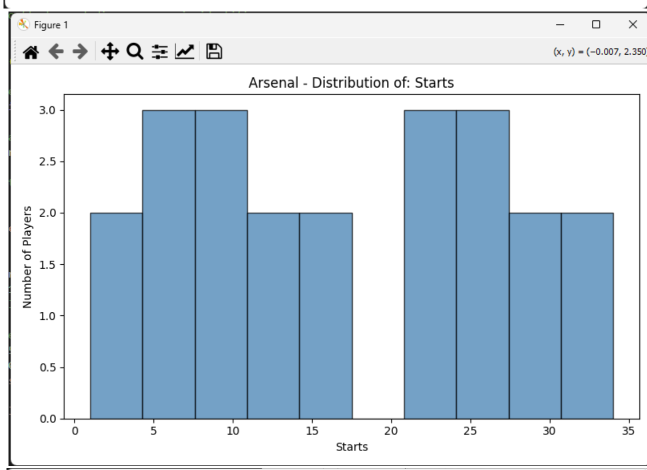
```
# For each team, generate a histogram for every numeric statistic
for team in teams:
    team_df = df[df['Squad'] == team]
    print(f"Team: {team} - Number of players: {len(team_df)}")

    for col in numeric_columns:
        values = team_df[col].dropna()
        if values.nunique() <= 1:
            continue  # Skip columns with no variation in data

        # Plot histogram for the current team's statistic
        plt.figure(figsize=(8, 5))
        plt.hist(values, bins=10, alpha=0.75, color='steelblue', edgecolor='black')
        plt.title(f"{team} - Distribution of: {col}")
        plt.xlabel(col)
        plt.ylabel("Number of Players")
        plt.tight_layout()
        plt.show()
```

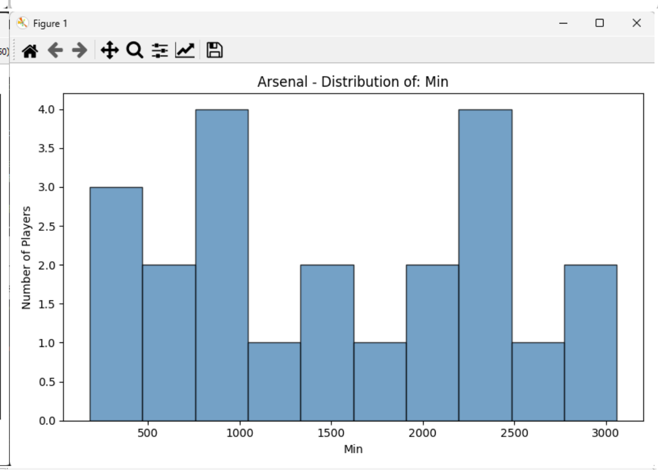
3.1.4 Output

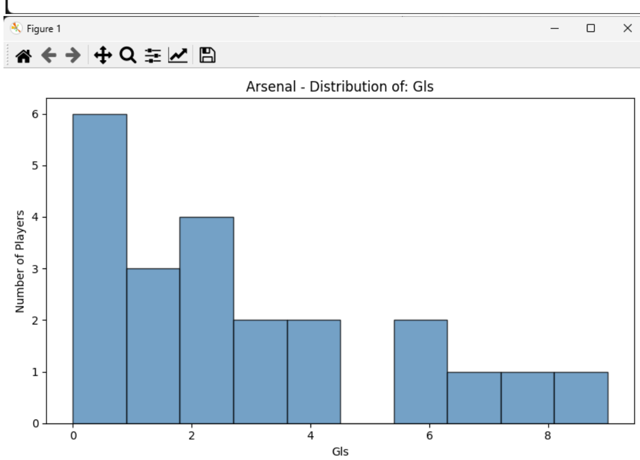
- The output consists of histograms displaying the distribution of each statistic for each team, color-coded by team for clear visualization (e.g., the first 6 stats of the Arsenal squad).

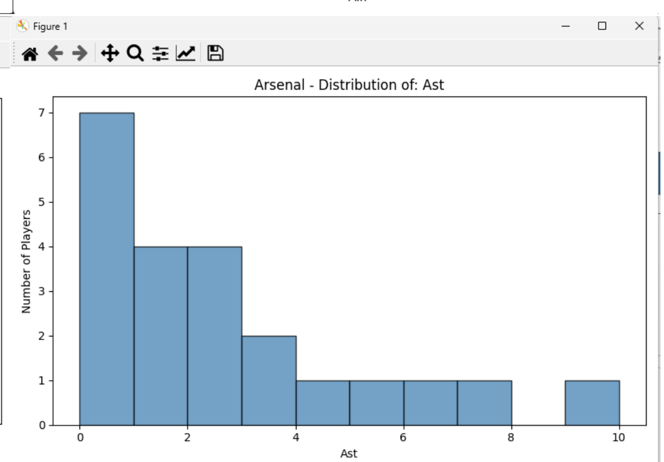












3.2 For all players in the league (Ve.py)

3.2.1 Objective Overview

- The script processes football player data and generates histograms and bar charts to visualize player statistics and distributions across positions.

3.2.2 Technology Description

- pandas: Used for loading, cleaning, and manipulating the dataset e ciently.
- matplotlib.pyplot: Used to create histograms and bar charts to visualize the distribution of player statistics and positions.
- csv, sys: These libraries handle CSV ﬁle processing and system-level constraints for large data handling.

```
import pandas as pd
import matplotlib.pyplot as plt
```

3.2.3 Code Breakdown

- The code starts by loading the results.csv dataset using pandas.

```
# Load the dataset from 'results.csv'
df = pd.read_csv("results.csv")
```

- Non-numeric values such as "N/a" are replaced with 0.00, and columns are converted to ﬂoat types.

```
# Replacing missing values with 0.00
df.replace("N/a", 0.00, inplace=True)
```

- The Age column is normalized from a "years-days" format to a decimal format (e.g., 20-200 → 20.55).

```
# Function to normalize age from 'years-days' format to float
def normalize_age(age_str):
    try:
        if isinstance(age_str, str) and '-' in age_str:
            years, days = map(int, age_str.split('-'))
            return round(years + days / 365, 2)
        return float(age_str)
    except:
        return 0.0

# Apply age normalization if 'Age' column exists
if 'Age' in df.columns:
    print("Normalizing 'Age' column ...")
    df['Age'] = df['Age'].apply(normalize_age)
```

- For each numeric column (excluding identiﬁers), a histogram is plotted, representing the distribution of values.

```
# Generate histograms for all numeric columns
print("Generating histograms for statistics ...")
for col in df.columns:
    if col in skip_headers:
        continue
    try:
        df[col] = pd.to_numeric(df[col], errors='coerce').fillna(0)
        print(f"Plotting: {col}")
        plt.hist(df[col], bins=30, color='skyblue', edgecolor='black')
        plt.title(f"Histogram: {col}")
        plt.xlabel(col)
        plt.ylabel("Number of Players")
        plt.tight_layout()
        plt.show()
    except Exception as e:
        # Skip columns that cause errors during numeric conversion or plotting
        print(f"Skipping column {col} due to error: {e}")
```

- Additionally, a bar chart is created to display the number of players by position, based on the 'Pos' column.

```
# Bar chart for number of players by position
if 'Pos' in df.columns:
    print("Plotting: Number of players by position")
    pos_counts = df['Pos'].value_counts()
    plt.bar(pos_counts.index, pos_counts.values, color='orange', edgecolor='black')
    plt.title("Number of Players by Position")
    plt.xlabel("Position")
    plt.ylabel("Number of Players")
    plt.tight_layout()
    plt.show()

# Process complete
print("Chart generation complete.")
```

- Any columns that cause errors during conversion or plotting are skipped.

3.2.4 Output

- The output consists of histograms displaying the distribution of each statistic for each player, color-coded by team for clear visualization (e.g., the ﬁrst 9 stats of all players).

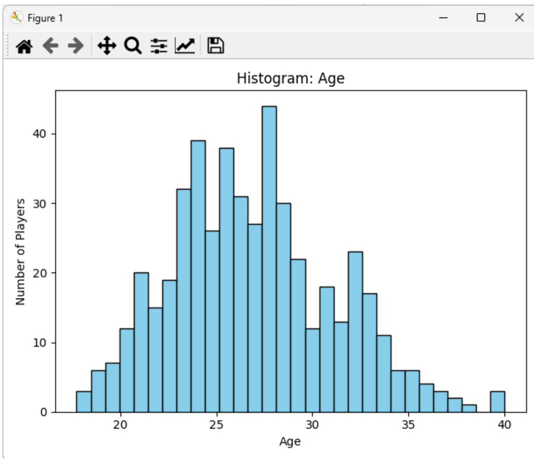


Figure 1
Histogram: Age
Number of Players
Age

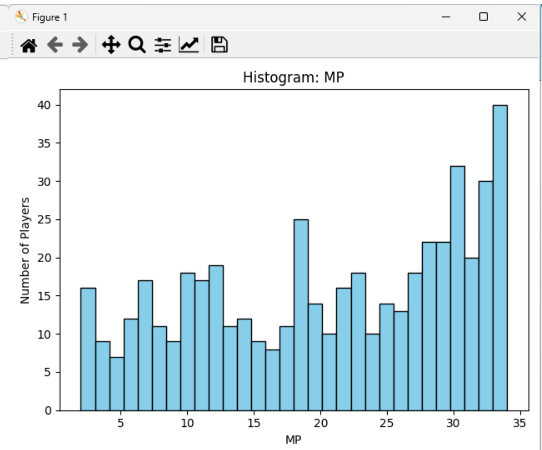


Figure 1
Histogram: MP
Number of Players
MP

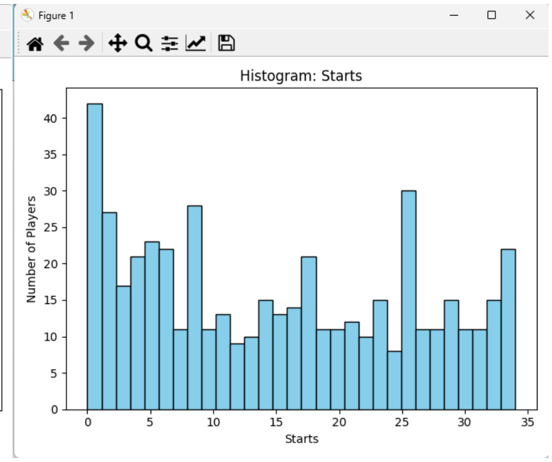


Figure 1
Histogram: Starts
Number of Players
Starts

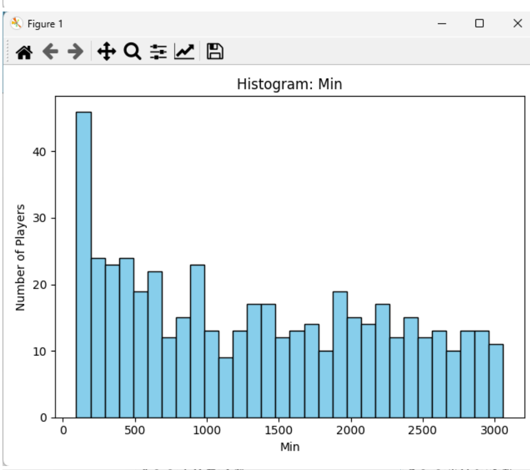


Figure 1
Histogram: Min
Number of Players
Min

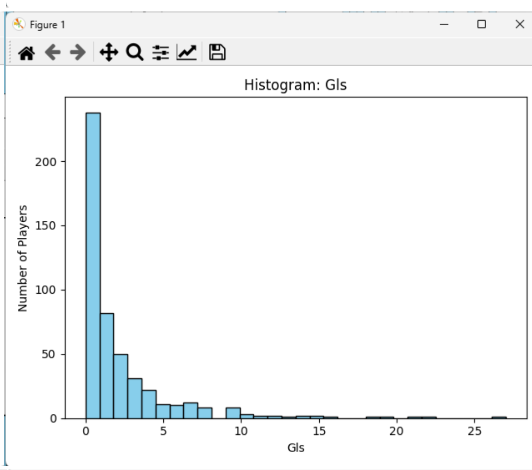


Figure 1
Histogram: Gls
Number of Players
Gls

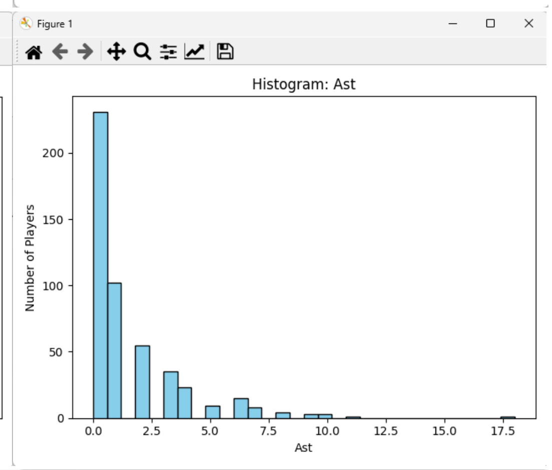


Figure 1
Histogram: Ast
Number of Players
Ast

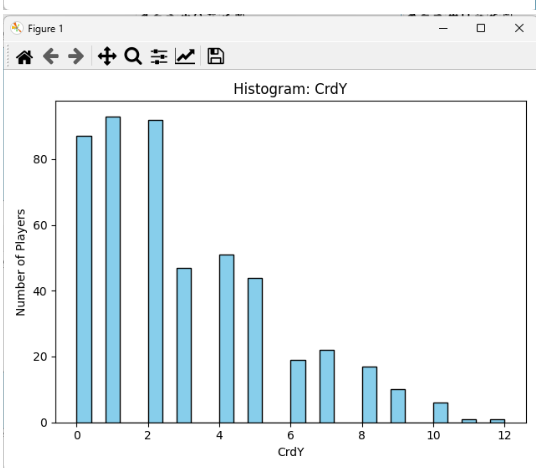


Figure 1
Histogram: CrdY
Number of Players
CrdY

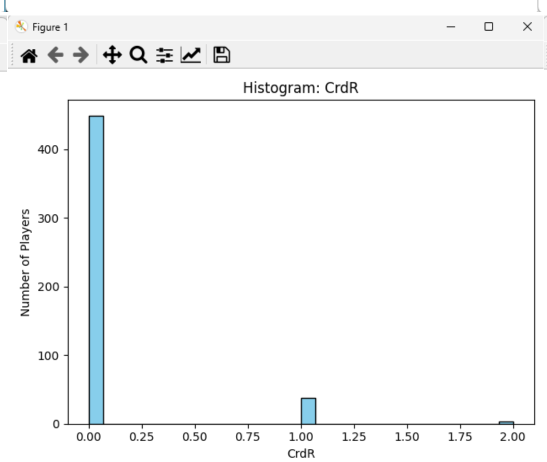


Figure 1
Histogram: CrdR
Number of Players
CrdR

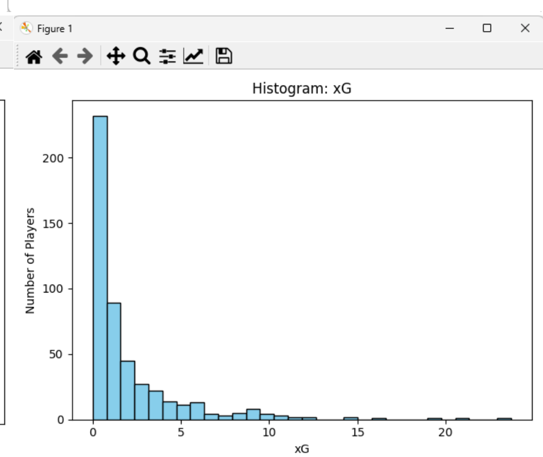


Figure 1
Histogram: xG
Number of Players
xG

4. Best Performing Team in the 2024-2025 Premier League season (DoiThanhTichTotNhat.py)

4.1 Objective Overview

- Build a prediction system to determine the champion team based on season-long player statistics.

4.2 Technology Description

- pandas: Used for reading, cleaning, grouping, and normalizing tabular data from the CSV file.
- tkinter: Provides GUI windows for displaying team logos and result messages.
- Pillow (PIL): Loads and resizes team logo images for display.
- os: Checks for the existence of image files (e.g., PNG or JPG logos).

```
import pandas as pd
import tkinter as tk
from PIL import Image, ImageTk
import os
```

4.3 Code Breakdown

- Shows initial playful popups with messages for certain teams to create anticipation.

```
def show_popup(title, message, image_path=None, wait_ms=5000):
    popup = tk.Tk()
    popup.title(title)
    popup.geometry("520x550")
    popup.configure(bg="white")
    popup.after(wait_ms, popup.destroy)

    frame = tk.Frame(popup, bg="white")
    frame.pack(expand=True, fill="both")

    if image_path and os.path.exists(image_path):
        img = Image.open(image_path).resize((300, 300))
        logo = ImageTk.PhotoImage(img)
        logo_label = tk.Label(frame, image=logo, bg="white")
        logo_label.image = logo
        logo_label.pack(pady=15)
    else:
        tk.Label(frame, text="(Logo not found)", font=("Arial", 12), bg="white").pack(pady=15)

    label = tk.Label(frame, text=message, font=("Arial", 16, "bold"), bg="white")
    label.pack(pady=15)

    popup.mainloop()
```

- Reads data from results.csv, replaces "N/a" with 0.0, and fills any missing values with 0.0.

```python
df = pd.read_csv("results.csv")
df.replace("N/a", 0.0, inplace=True)
df.fillna(0.0, inplace=True)
```

- Two main stat categories: attack_cols (e.g., Gls, Ast, xG) and defense_cols (e.g., Tkl, Blocks, Save%).

```python
attack_cols = ["Gls", "Ast", "xG", "xAG", "PrgPs", "SCA", "GCA"]
defense_cols = ["Tkl", "Blocks", "Int", "GA90", "Save%", "CS%"]
all_stat_cols = attack_cols + defense_cols
```

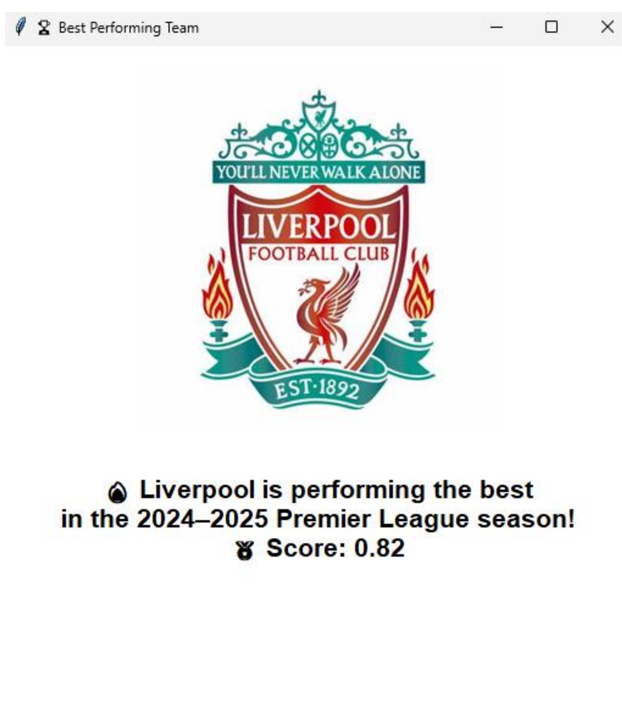
- Averages player stats per team (groupby("Squad")).
- Normalizes all stats to a [0, 1] scale.
- Calculates a TeamScore using: TeamScore = 0.6 × Offensive Mean + 0.4 × Defensive Mean

```python
team_stats["TeamScore"] = (
    team_stats[attack_cols].mean(axis=1) * 0.6 +
    team_stats[defense_cols].mean(axis=1) * 0.4
)
```

- Identifies the team with the highest TeamScore

4.4 Output

- A final GUI popup presents the predicted champion with the team logo and overall score.



5. Representative O}ensive and Defensive Statistics(ATK_DF.py)

5.1 Objective Overview

- Objective: The goal is to visualize the distribution of speci%c football player statistics for attack and defense.
- Selected Metrics: We are focusing on three attacking metrics: Goals scored (Gls), Assists (Ast), and Expected Goals (xG); and three defensive metrics: Tackles (Tkl), Interceptions (Int), and Blocks (Blocks).

5.2 Technology Description

- pandas: This library is used to load and process structured CSV data, such as football player statistics.
- matplotlib: A visualization tool to generate histograms, helping to understand the distribution of the selected statistics.

```
import pandas as pd
import matplotlib.pyplot as plt
```

5.3 Code Breakdown

- The dataset results.csv is read into a DataFrame.
- Goalkeepers (Pos == 'GK') are removed from the dataset, focusing on out%eld players.

```
# Read data
df = pd.read_csv('results.csv')
df = df[df['Pos'] != 'GK']  # Remove goalkeepers
```

- Attack Metrics: Goals (Gls), Assists (Ast), and Expected Goals (xG).
- Defense Metrics: Tackles (Tkl), Interceptions (Int), and Blocks (Blocks).

```
# Select 3 representative attack and defense metrics
attack_columns = ['Gls', 'Ast', 'xG']
defense_columns = ['Tkl', 'Int', 'Blocks']
```

- A function plot_histogram_for_column is created to generate a histogram for a given column, showing the distribution of player statistics.

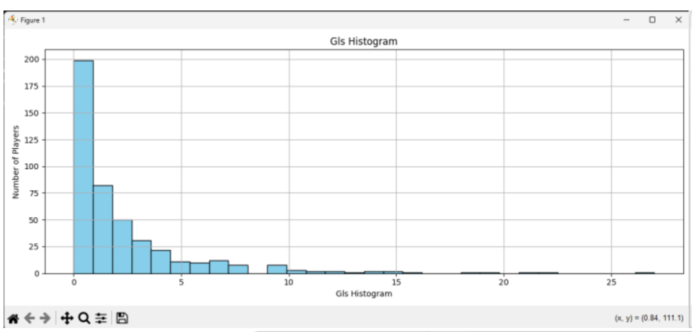
```python
# Function to plot histograms for each statistic
def plot_histogram_for_column(df, column, title, color):
    plt.figure(figsize=(12, 5))
    plt.hist(df[column], bins=30, color=color, edgecolor='black')
    plt.title(title)
    plt.xlabel(f'{title}')
    plt.ylabel('Number of Players')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

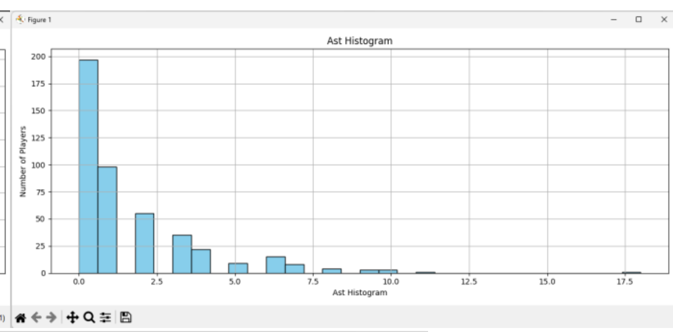
- Histograms are plotted for each of the selected metrics (both attacking and defensive).

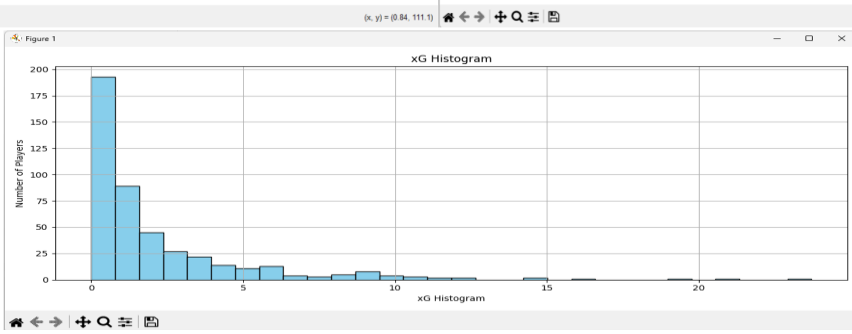
```python
# Plot histograms for selected attacking metrics
for col in attack_columns:
    plot_histogram_for_column(df, col, f'{col} Histogram', 'skyblue')

# Plot histograms for selected defensive metrics
for col in defense_columns:
    plot_histogram_for_column(df, col, f'{col} Histogram', 'lightgreen')
```

5.4 Output

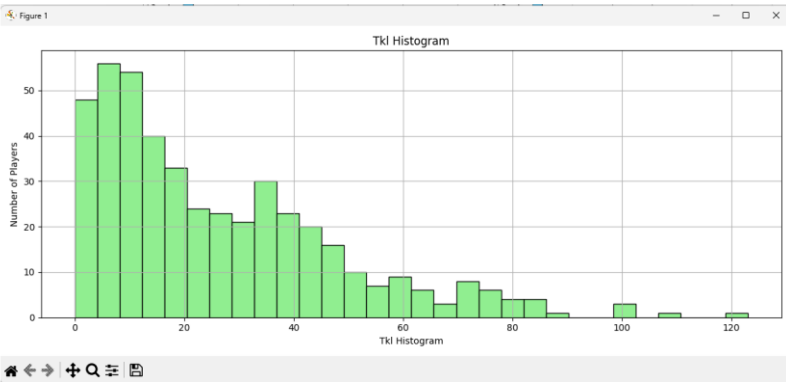
- Histograms for Attack Metrics:
    - Goals (Gls), Assists (Ast), and Expected Goals (xG).

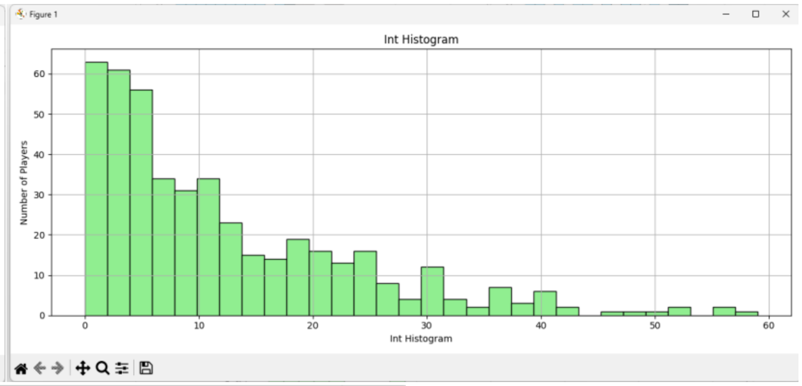


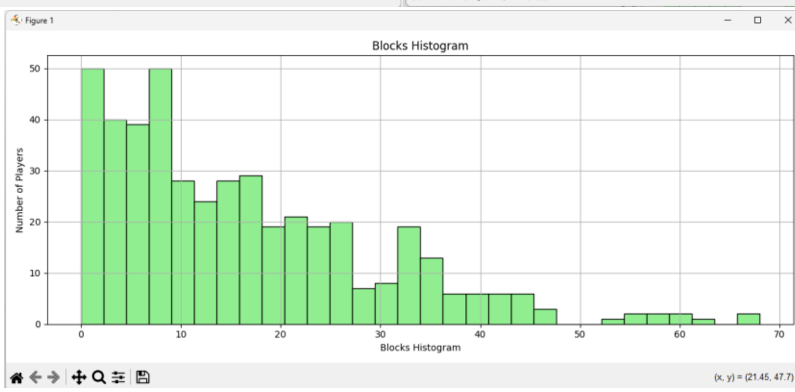




- Histograms for Defense Metrics:
    - Tackles (Tkl), Interceptions (Int), and Blocks (Blocks).







Problem III

1. K-means algorithm (TimK.py)

1.1 Find K (TimK.py)

1.1.1 Objective Overview

- Use K-Means clustering to explore patterns in football player statistics.
- Determine the optimal number of clusters using the Elbow Method.

1.1.2 Technology Description

- pandas: Loads and processes structured CSV data.
- matplotlib: Visualizes the Elbow Method to help select the best number of clusters.
- scikit-learn:
  - StandardScaler: Standardizes numerical data.
  - KMeans: Applies clustering to group players with similar statistical pro%les.

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

1.1.3 Code Breakdown

- Loads results.csv.

```python
# Load the dataset
df = pd.read_csv("results.csv")
```

- Selects all columns starting from 'Nation' onward for clustering

```python
# Extract statistics starting from the 'Nation' column onward
stats = df.loc[:, 'Nation':]
```

- Removes percentage signs (%) from values.
- Converts strings to numeric values, coercing errors to NaN.
- Drops columns with all missing values, and fills the rest with 0.00.

```python
# Remove percentage symbols and convert to numeric values
stats = stats.apply(lambda col: col.astype(str).str.replace('%', '', regex=False))
stats = stats.apply(pd.to_numeric, errors='coerce')
stats = stats.dropna(axis=1, how='all')
stats.fillna(0.00, inplace=True)
```

- Uses StandardScaler to normalize feature values to zero mean and unit variance.

```python
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(stats)
```

- Tests multiple-cluster counts K from 2 to 77.

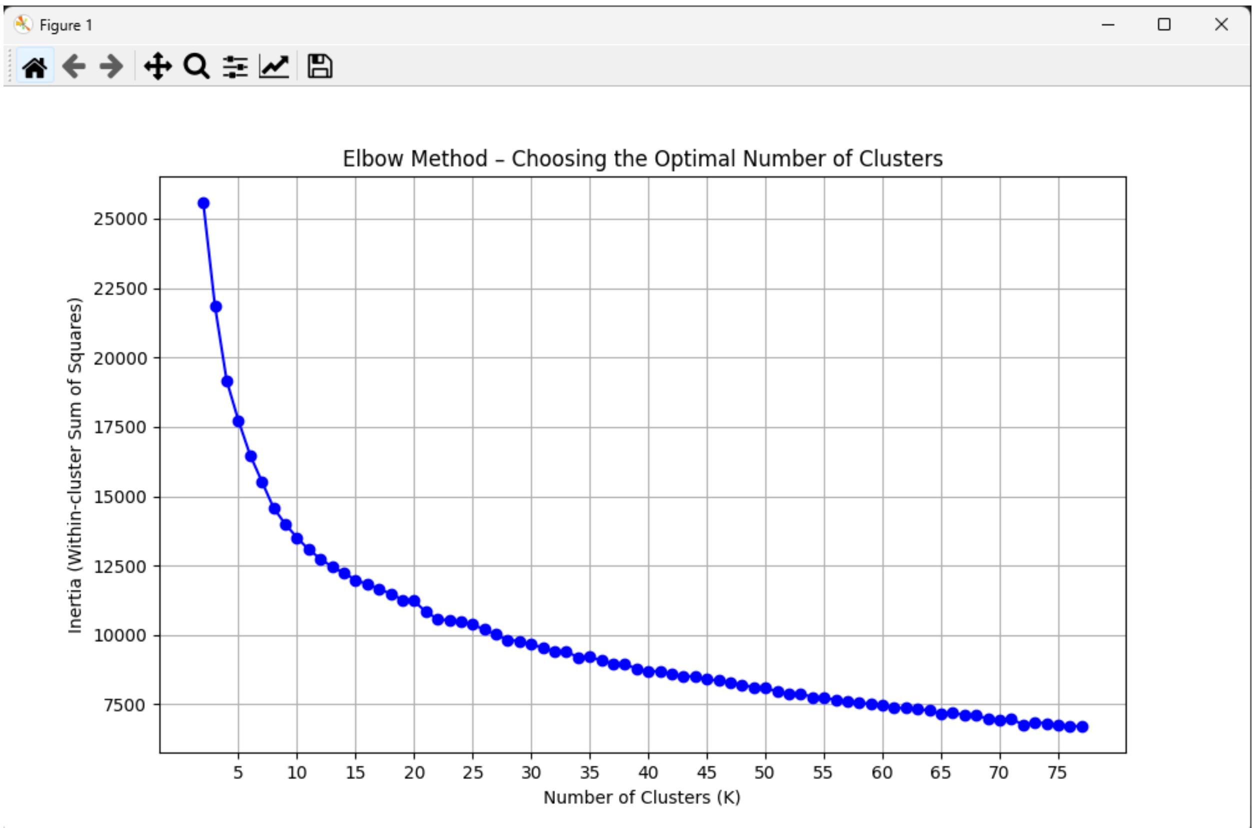
```python
# Evaluate inertia for a range of cluster numbers
inertia = []
K = range(2, 78)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

- For each K, calculates inertia (sum of squared distances within clusters).
- Plots inertia vs. number of clusters to identify the "elbow" point, which suggests the optimal number of clusters.

```python
# Plot the Elbow method result
plt.figure(figsize=(10, 6))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Within-cluster Sum of Squares)')
plt.title('Elbow Method – Choosing the Optimal Number of Clusters')
plt.grid(True)
plt.xticks(range(5, 80, 5))
plt.show()
```

1.1.4 Output

- A plot titled Elbow Method Choosing the Optimal Number of Clusters .
- Helps visually identify the ideal K where inertia stops decreasing signi%cantly.



1.1.5 Answer the question

- How many groups should the players be classi%ed into? Why?
  - According to my observation on the chart, K = 10 is a reasonable choice because it represents the "elbow point" where the rate of decrease in within-cluster variance (inertia) signi%cantly slows down.
  - Because diminishing returns after K = 10:
    - Before K = 10, each additional cluster signi%cantly improves how well the model %ts the data (by reducing inertia).

        - After K = 10, the improvement becomes small meaning that adding more clusters doesn t bring much value, but increases complexity.

- Provide your comments on the results.
    - The clustering with K = 10 produced ten distinct groups of football players based on a wide range of performance and positional statistics. This number of clusters strikes a balance between underfitting and over-segmentation, and is supported by the Elbow Method.
    - Each group likely reflects different player profiles or roles, such as:
        - High-scoring forwards
        - Creative midfielders
        - Defensive midfielders
        - Ball-playing defenders
        - Traditional centre-backs
        - Full-backs or wing-backs
        - Goalkeepers
        - Impact substitutes
        - Young emerging players
        - Veteran players with specialized roles
    - Strengths:
        - Clear differentiation between player types
    - Limitations:
        - Some overlap might occur if players play hybrid roles (e.g., defender-midfielder).

2. Plot a 2D cluster of the data points (PhanLoaiCauThu.py)

2.1 Objective Overview

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

2.2 Technology Description

- Pandas: Used to load and preprocess the dataset.
- Plotly: For interactive and dynamic scatter plot visualization of clustering results.

- Matplotlib: Used for Elbow Method to determine the optimal number of clusters.

- Plotly: For interactive and dynamic scatter plot visualization of clustering results.
- Scikit-learn:
    - StandardScaler for data normalization.
    - KMeans for unsupervised clustering.
    - PCA for dimensionality reduction and visualization.

```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import plotly.express as px
```

2.3 Code Breakdown

- Read data from results.csv.

```python
# Load dataset
df = pd.read_csv("results.csv")
```

- Keep statistical columns starting from 'Nation' onward.

```python
# Extract stats from 'Nation' to the end
stats = df.loc[:, 'Nation':]
```

- Remove % symbols and convert all data to numeric.

```python
# Clean percentage values and convert to numeric
stats = stats.apply(lambda col: col.astype(str).str.replace('%', '', regex=False))
stats = stats.apply(pd.to_numeric, errors='coerce')
```

- Fill any missing values with 0.

```python
# Fill missing values with 0
stats.fillna(0, inplace=True)
```

- Apply StandardScaler to normalize all features a key step before clustering.

```python
# Apply K-means clustering
k = 10
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
labels = kmeans.fit_predict(X_scaled)
df['Cluster'] = labels + 1  # Cluster starts from 1
```

- Perform K-means clustering with k=10.
- Use PCA to reduce the high-dimensional data to 2 components for visualization.
- Store PCA coordinates in new columns 'PC1' and 'PC2'.
- Use Plotly Express to plot a scatter plot of PC1 vs PC2.
- Color-code points by cluster.

- Add hover info: Player name, squad, position, and age.

```python
# Reduce dimensionality for visualization
pca = PCA(n_components=2)
pca_stats = pca.fit_transform(X_scaled)
df['PC1'] = pca_stats[:, 0]
df['PC2'] = pca_stats[:, 1]

# Plot PCA scatter plot with clusters
fig = px.scatter(
    df,
    x='PC1',
    y='PC2',
    color='Cluster',
    hover_data=['Player', 'Squad', 'Pos', 'Age'],
    title=f"K-means Clustering (K={k}, PCA Projection)",
    color_continuous_scale='Viridis'
)

# Customize marker size and layout
fig.update_traces(marker=dict(size=8, opacity=0.7))
fig.update_layout(
    xaxis_title="Principal Component 1",
    yaxis_title="Principal Component 2",
    legend_title="Cluster"
)

# Show the interactive scatter plot
fig.show()
```

2.4 Output

- An interactive 2D scatter plot where players are grouped into clusters (1 10) with di}erent colors. Hovering over points shows player details, helping assess clustering e}ectiveness and group patterns.

Problem IV

1. Collect the transfer values of players for the 2024-2025 season from https://www.footballtransfers.com/ whose playing time exceeds 900 minutes.

1.1 Claw Data (LayData.py)

1.1.1 Objective Overview

- The script scrapes the list of most valuable Premier League players from the FootballTransfers website, collecting player names, teams, and stats across multiple pages. The data is then saved to a CSV ﬁle, with player and team names normalized and unnecessary columns removed.

1.1.2 Technology Description

- Selenium: Used for automating web browsing tasks to access and scrape data from the website. The script utilizes the Chrome browser in headless mode for e ciency.
- BeautifulSoup: A Python library used for parsing HTML and extracting the relevant data from the web page.
- Pandas: Used to store and manipulate the data collected from the website before saving it as a CSV ﬁle.
- Webdriver Manager: Manages the ChromeDriver installation and ensures compatibility between the Selenium version and the browser.

```
import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from bs4 import BeautifulSoup
import time
```

- Custom Module (XuLyData.py): Contains utility functions for data cleaning, column renaming, and header normalization.

  - def XoaHeader(): This function removes one or more columns from a dataset based on the column names passed to it.

```python
# Function to remove specified columns from the dataset
def XoaHeader(columns_to_remove, input_file, output_file):
    df = pd.read_csv(input_file)
    print("📌 Before removal:", df.columns.tolist())

    # Loop through columns to remove and drop them if they exist
    for col in columns_to_remove:
        if col in df.columns:
            df.drop(columns=col, inplace=True)
            print(f"🗑 Removed column: {col}")
        else:
            print(f"⚠ Column not found: {col}")

    # Save the modified dataset to a new CSV file
    df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"✅ Saved the new file to: {output_file}")
```

  - def DoiTen(): This function renames a column in the dataset.

```python
# Function to rename a column in the dataset
def DoiTen(clo_1, clo_2, input_file, output_file):
    df = pd.read_csv(input_file)
    print("📌 Before renaming:", df.columns.tolist())

    if clo_1 in df.columns:
        df.rename(columns={clo_1: clo_2}, inplace=True)
        print(f"📝 Renamed column '{clo_1}' to '{clo_2}'")
    else:
        print(f"⚠ Column not found: {clo_1}")

    # Save the modified dataset to a new CSV file
    df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"✅ Saved the new file to: {output_file}")
```

- def ChuanHoaTen(): This function normalizes player names by removing any reference to the squad name from the player name column.

```python
# Function to normalize player names by removing squad names from the player column
def ChuanHoaTen(input_file, output_file):
    df = pd.read_csv(input_file)
    print("📌 Before normalizing Player:", df[['Player', 'Squad']].head())

    if 'Player' in df.columns and 'Squad' in df.columns:
        # Strip squad name from the player name if found
        df['Player'] = df.apply(lambda row: str(row['Player']).split(str(row['Squad']))[0].strip() if str(row['Squad']) in str(row['Player']) else str(row['Player']).strip(), axis=1)
        print("✅ Removed Squad from Player")
    else:
        print("⚠️ 'Player' or 'Squad' column not found in data")

    # Save the modified dataset to a new CSV file
    df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"✅ Saved the data to: {output_file}")
```

- def ChuanHoaTen2(): This function further normalizes player names by identifying and removing any repeated characters in the player names.

```python
# Function to further normalize player names by removing repeated characters
def ChuanHoaTen2(input_file, output_file):
    df = pd.read_csv(input_file)
    print("📌 Before normalizing Player:", df['Player'].head())

    if 'Player' in df.columns:
        def ThoiTaChiaDoi(s):
            s = str(s).strip()
            length = len(s)
            for i in range(1, length // 2 + 1):
                if length % i == 0:
                    substring = s[:i]
                    if substring * (length // i) == s:
                        return substring
            return s

        # Apply the function to remove repeating characters in player names
        df['Player'] = df['Player'].apply(ThoiTaChiaDoi)
        print("✅ Removed repeating parts in Player")
    else:
        print("⚠️ 'Player' column not found in data")

    # Save the modified dataset to a new CSV file
    df.to_csv(output_file, index=False, encoding='utf-8-sig')
    print(f"✅ Saved the data to: {output_file}")
```

- def ChuanHoaSquad: This function normalizes squad names based on a predefined mapping. It updates squad names that may have variations or abbreviations.

```python
# Function to normalize squad names according to a predefined mapping
def ChuanHoaSquad(input_file, output_file):
    df = pd.read_csv(input_file)
    print("📌 Before normalizing Squad:", df['Squad'].unique())

    if 'Squad' in df.columns:
        SquadMapping = {
            'Arsenal': 'Arsenal',
            'Aston Villa': 'Aston Villa',
            "B'mouth": 'Bournemouth',
            'Brentford': 'Brentford',
            'Brighton': 'Brighton',
            'Chelsea': 'Chelsea',
            'C. Palace': 'Crystal Palace',
            'Everton': 'Everton',
            'Fulham': 'Fulham',
            'Ipswich Town': 'Ipswich Town',
            'Liverpool': 'Liverpool',
            'Leicester': 'Leicester City',
            'Man Utd': 'Manchester Utd',
            'Man City': 'Manchester City',
            'Newcastle Utd.': 'Newcastle Utd',
            'Nottingham': "Nott'ham Forest",
            'Southampton': 'Southampton',
            'Tottenham': 'Tottenham',
            'West Ham United': 'West Ham',
            'Wolverhampton': 'Wolves'
        }

        # Apply the mapping to normalize the squad names
        df['Squad'] = df['Squad'].apply(lambda x: SquadMapping.get(str(x).strip(), x))
        print("✅ Normalized Squad names")
    else:
        print("⚠️ 'Squad' column not found in data")
```

1.1.3 Code Breakdown

- The Chrome browser is configured in headless mode using Selenium.
- Webdriver Manager installs the correct ChromeDriver.

```python
# Configure the Chrome browser for Selenium
options = webdriver.ChromeOptions()
options.add_argument("--headless=new")  # Run browser in headless mode (without GUI)
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
```

- The script loops through 22 pages, fetching data related to the players' names, teams, and stats from the player table (#player-table-body).

```python
# Loop through pages of data on the website
for page in range(1, 23):
    url = f"https://www.footballtransfers.com/us/values/players/most-valuable-soccer-players/playing-in-uk-premier-league/{page}"
    success = False

    # Retry up to 3 times if there's an error while collecting data
    for attempt in range(3):
        try:
            # Access the page and scroll down to load the entire content
            driver.get(url)
            time.sleep(2)
            driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
            time.sleep(2)

            # Wait until the player table appears
            WebDriverWait(driver, 10).until(
                EC.presence_of_element_located((By.ID, "player-table-body"))
            )
```

- It waits for the table to load fully and handles retries in case of errors.
- The player's name, team, and stats are extracted from the HTML table and stored in a list.
- After extraction, the column names are captured and saved for later use

```python
# Extract the data rows from the table
tbody = soup.find('tbody', id='player-table-body')
rows_added = 0
if tbody:
    for row in tbody.find_all('tr'):
        cols = row.find_all('td')
        if len(cols) < 2:  # Skip rows that don't have enough data
            continue

        # Get the player's name and team
        a_player = cols[0].find('a')
        player = a_player.get_text(strip=True) if a_player else cols[0].get_text(strip=True)

        a_team = cols[1].find('a')
        team = a_team.get_text(strip=True) if a_team else cols[1].get_text(strip=True)

        # Extract other data in the row
        other_data = [c.get_text(strip=True) for c in cols[2:]]
        row_data = [player, team] + other_data

        # If the row contains valid data, add it to the list
        if any(cell for cell in row_data):
            all_rows.append(row_data)
            rows_added += 1
```

- The collected data is saved into a Pandas DataFrame and then written to a CSV file (GiaTriCauThu.csv).

```
# Save all the collected data into a CSV file
df = pd.DataFrame(all_rows, columns=column_names)
df.to_csv("GiaTriCauThu.csv", index=False, encoding='utf-8-sig')
print(f"✅ Total: {len(df)} rows have been saved to 'GiaTriCauThu.csv'")
```

- Additional steps are taken to clean and normalize the data, such as renaming columns, removing unnecessary ones, and standardizing player and team names using custom functions.

```
#  Import and process data AFTER CSV file has been created
import XuLyData as XL
XL.XoaHeader(["Skill/ pot"], "GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Remove 'Skill/ pot' column
XL.DoiTen("#", "STT", "GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Rename '#' column to 'STT'
XL.DoiTen("Team", "Squad", "GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Rename 'Team' column to 'Squad'
XL.ChuanHoaTen("GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Normalize player names
XL.ChuanHoaTen2("GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Normalize player names again
XL.ChuanHoaSquad("GiaTriCauThu.csv", "GiaTriCauThu.csv")  # Normalize squad/team names
```

1.1.4 Output

- The script processes and cleans GiaTriCauThu.csv, which contains Premier League player data including names, teams, and stats. It renames columns and normalizes player and team names, preparing the data for analysis or modeling.

```
STT, Player                 , Age, Squad           , ETV
  1, Erling Haaland         ,  24, Manchester City, €199.6M
  2, Martin Ødegaard        ,  26, Arsenal         , €125.8M
  3, Alexander Isak         ,  25, Newcastle Utd   , €119.4M
  4, Cole Palmer            ,  22, Chelsea         , €117.4M
  5, Alexis Mac Allister    ,  26, Liverpool       , €117M
  6, Declan Rice            ,  26, Arsenal         , €116.4M
  7, Bukayo Saka            ,  23, Arsenal         , €113M
  8, Phil Foden             ,  24, Manchester City, €99.8M
  9, Rodri                  ,  28, Manchester City, €87M
 10, Ryan Gravenberch       ,  22, Liverpool       , €85.5M
 11, Moisés Caicedo         ,  23, Chelsea         , €82.8M
 12, Bruno Guimarães        ,  27, Newcastle Utd   , €82.3M
 13, Morgan Rogers          ,  23, Aston Villa     , €79.6M
 14, William Saliba         ,  24, Arsenal         , €79.5M
 15, Omar Marmoush          ,  26, Manchester City, €79.1M
 16, Joško Gvardiol         ,  23, Manchester City, €77.4M
 17, Sávio                  ,  21, Manchester City, €74.8M
 18, Rúben Dias             ,  27, Manchester City, €74.5M
 19, Enzo Fernández         ,  24, Chelsea         , €74.2M
 20, Lucas Paquetá          ,  27, West Ham        , €73.5M
 21, Archie Gray            ,  19, Tottenham       , €72.5M
 22, Dominik Szoboszlai     ,  24, Liverpool       , €70.5M
 23, Kai Havertz            ,  25, Arsenal         , €70M
 24, Luis Díaz              ,  28, Liverpool       , €69.3M
 25, Cody Gakpo             ,  25, Liverpool       , €67.7M
 26, Brennan Johnson        ,  23, Tottenham       , €67M
 27, Nicolas Jackson        ,  23, Chelsea         , €66.9M
 28, Gabriel Magalhães      ,  27, Arsenal         , €66.5M
 29, Jérémy Doku            ,  22, Manchester City, €66.5M
 30, Leny Yoro              ,  19, Manchester Utd  , €66.2M
 31, Murillo                ,  22, Nott'ham Forest, €66M
 32, Amadou Onana           ,  23, Aston Villa     , €64.4M
 33, Ollie Watkins          ,  29, Aston Villa     , €63.2M
 34, Mohammed Kudus         ,  24, West Ham        , €63M
 35, Jurrien Timber         ,  23, Arsenal         , €62.7M
 36, Rico Lewis             ,  20, Manchester City, €62.7M
 37, Matheus Cunha          ,  25, Wolves          , €62.6M
```

1.2 Merge Data (ETV.py & ETV2.py)

ETV

1.2.1 Objective Overview

1.2.2 Technology Description

- Pandas: Used for reading, cleaning, normalizing, and merging CSV data e ciently.

```python
import pandas as pd
```

1.2.3 Code Breakdown

- Reads data from GiaTriCauThu.csv (ETV) and ChoiTren900p.csv (playing time >900 minutes).

```python
# Read data from CSV files
df_transfers = pd.read_csv("GiaTriCauThu.csv", encoding='utf-8-sig')
df_played = pd.read_csv("ChoiTren900p.csv", on_bad_lines='skip')
```

- Cleans column names by stripping leading and trailing spaces.

```python
# Clean column names
df_transfers.columns = df_transfers.columns.str.strip()
df_played.columns = df_played.columns.str.strip()
```

- Checks for the existence of the 'ETV' column in GiaTriCauThu.csv.

```python
# Check if 'ETV' column exists in df_transfers
if 'ETV' not in df_transfers.columns:
    raise ValueError("ETV column not found in GiaTriCauThu.csv")
```

- Merges the datasets (df_played and df_transfers) on the 'Player' column, keeping only the matched players and their ETV values.

```python
# Merge datasets without modifying player names
merged = pd.merge(
    df_played[['Player', 'Squad']],
    df_transfers[['Player', 'ETV']],
    on='Player',  # Matching on exact 'Player' names
    how='left'
)
```

- Identifies and prints the list of players that did not match (missing ETV values).
- Saves unmatched players (missing ETV) to Thieu.csv.

```python
# Separate players missing ETV
missing_etv = merged[merged['ETV'].isna()]
if not missing_etv.empty:
    # Drop the 'ETV' column from the missing players
    missing_etv = missing_etv.drop(columns=['ETV'])
    # Save missing players to 'Thieu.csv' without the 'ETV' column
    missing_etv.to_csv("Thieu.csv", index=False, encoding='utf-8-sig')
    print(f"⚠ {len(missing_etv)} players are missing ETV values. Saved to Thieu.csv without ETV column.")
```

- Filters out players without an ETV value and adds a sequential index.
- Saves the processed data with ETV to ETV.csv.

```python
# Save result to 'ETV.csv'
result.to_csv("ETV.csv", index=False, encoding='utf-8-sig')
print(f"✅ Processed data for {len(result)} players with ETV.")
print("First 5 rows of the result:")
print(result.head())
```

1.2.4 Output

- ETV.csv: List of players with >900 minutes and their corresponding ETV values.

```
BTLPython > Bai4 > Phan1 > ETV.csv > data
STT,Player,Squad,ETV
1,Aaron Ramsdale,Southampton,€18.7M
2,Aaron Wan-Bissaka,West Ham,€26.9M
3,Abdoulaye Doucouré,Everton,€5.8M
4,Adam Smith,Bournemouth,€1.5M
5,Adam Wharton,Crystal Palace,€48.9M
6,Adama Traoré,Fulham,€8M
7,Alejandro Garnacho,Manchester Utd,€60.7M
8,Alex Iwobi,Fulham,€30.3M
9,Alexander Isak,Newcastle Utd,€120.3M
10,Alexis Mac Allister,Liverpool,€106.1M
11,Alisson,Liverpool,€24.1M
12,Alphonse Aréola,West Ham,€11.8M
13,Amad Diallo,Manchester Utd,€48.5M
14,Amadou Onana,Aston Villa,€62.1M
15,Andreas Pereira,Fulham,€19M
16,Andrew Robertson,Liverpool,€24M
17,André,Wolves,€29.8M
18,André Onana,Manchester Utd,€34.2M
19,Anthony Elanga,Nott'ham Forest,€45.5M
20,Anthony Gordon,Newcastle Utd,€49.5M
21,Antoine Semenyo,Bournemouth,€47.7M
22,Antonee Robinson,Fulham,€46.4M
23,Archie Gray,Tottenham,€58.2M
24,Arijanet Murić,Ipswich Town,€10.1M
25,Ashley Young,Everton,€1.3M
26,Axel Tuanzebe,Ipswich Town,€4.2M
27,Bart Verbruggen,Brighton,€30.7M
28,Ben Davies,Tottenham,€5.6M
29,Ben Johnson,Ipswich Town,€12.6M
30,Bernardo Silva,Manchester City,€50M
31,Bernd Leno,Fulham,€13.7M
32,Beto,Everton,€24.4M
33,Bilal El Khannouss,Leicester City,€40.6M
34,Boubacar Kamara,Aston Villa,€46.3M
35,Boubakary Soumaré,Leicester City,€15.4M
36,Brennan Johnson,Tottenham,€71.3M
37,Bruno Fernandes,Manchester Utd,€54.8M
38,Bruno Guimarães,Newcastle Utd,€83.2M
39,Bryan Mbeumo,Brentford,€57.2M
40,Bukayo Saka,Arsenal,€101.3M
41,Caleb Okoli,Leicester City,€16.7M
42,Callum Hudson-Odoi,Nott'ham Forest,€30.2M
43,Calvin Bassey,Fulham,€29.9M
44,Cameron Archer,Southampton,€18.6M
45,Cameron Burgess,Ipswich Town,€1.7M
46,Carlos Baleba,Brighton,€52.4M
47,Carlos Soler,West Ham,€23.4M
48,Casemiro,Manchester Utd,€11.6M
49,Chris Richards,Crystal Palace,€15.7M
50,Chris Wood,Nott'ham Forest,€7.8M
```

- Thieu.csv: List of players without ETV values (unmatched players).

```
Player,Squad
Adam Armstrong,Southampton
Alphonse Areola,West Ham
Arijanet Muric,Ipswich Town
Idrissa Gana Gueye,Everton
Igor,Brighton
Ismaila Sarr,Crystal Palace
Jeremy Doku,Manchester City
Jurriën Timber,Arsenal
Kyle Walker,Manchester City
Mads Roerslev,Brentford
Manuel Ugarte Ribeiro,Manchester Utd
Mario Lemina,Wolves
Milos Kerkez,Bournemouth
Omari Hutchinson,Ipswich Town
Radu Drăgușin,Tottenham
Rasmus Højlund,Manchester Utd
Rayan Aït-Nouri,Wolves
Son Heung-min,Tottenham
Victor Bernth Kristiansen,Leicester City
```

ETV2

1.2.5 Objective Overview

- The goal of this script is to generate a list of players by comparing and merging player data across different CSV files, and to finally generate an updated ETV.csv containing player names, squads, and their Estimated Transfer Values (ETV). The process involves creating a CheckLan2.csv file by subtracting the data in ETV.csv from GiaTriCauThu.csv, and using fuzzy string matching to match players between Thieu.csv and the newly created file.

1.2.6 Technology Description

- Pandas: Used extensively to read, filter, merge, and manipulate tabular data stored in CSV files.

- FuzzyWuzzy: Applies fuzzy string matching to identify approximate matches between player names that may differ slightly in spelling or formatting.
- Unidecode: Normalizes player names by removing accents and special characters, enabling more effective string comparisons.

```python
import pandas as pd
from fuzzywuzzy import fuzz
from unidecode import unidecode
import time
```

1.2.7 Code Breakdown

- Reads data from ETV.csv, GiaTriCauThu.csv, and Thieu.csv.

```python
# Read data
df_etv = pd.read_csv('ETV.csv')
df_gt = pd.read_csv('GiaTriCauThu.csv')
df_thieu = pd.read_csv('Thieu.csv')
```

- Cleans column names to remove any extra spaces.

```python
# Clean column names
df_etv.columns = df_etv.columns.str.strip()
df_gt.columns = df_gt.columns.str.strip()
df_thieu.columns = df_thieu.columns.str.strip()
```

- Generates CheckLan2.csv by finding players from GiaTriCauThu.csv that are not in ETV.csv based on the "Player" and "Squad" columns.

```python
# Generate CheckLan2.csv by subtracting ETV from GiaTriCauThu
df_checklan2 = pd.merge(df_gt, df_etv[['Player', 'Squad']], on=['Player', 'Squad'], how='left', indicator=True)
df_checklan2 = df_checklan2[df_checklan2['_merge'] == 'left_only'].drop(columns=['_merge'])
df_checklan2.to_csv('CheckLan2.csv', index=False, encoding='utf-8-sig')
```

- For each player in Thieu.csv, fuzzy matching is performed with players in CheckLan2.csv (using a score of 55 or higher) to find potential matches.

- Matches are then cross-checked to ensure that they don't already exist in ETV.csv

```python
# Matching process
matches = []
matched_players = set()
checked_players = {}

for index, thieu_row in df_thieu.iterrows():
    player_thieu = thieu_row['Player']
    squad_thieu = thieu_row['Squad']
    if player_thieu in matched_players:
        continue

    best_match = None
    for _, check_row in df_checklan2.iterrows():
        player_check = check_row['Player']
        squad_check = check_row['Squad']
        score = fuzz.partial_ratio(player_thieu.lower(), player_check.lower())

        if score >= 55 and squad_thieu == squad_check:
            # Check if already exists in df_etv
            already_exists = (
                ((df_etv['Player'] == player_check) & (df_etv['Squad'] == squad_check)).any()
            )
            if not already_exists:
                best_match = (player_check, squad_check, player_thieu, squad_thieu, score)
            break
        if score < 55 and player_thieu not i (variable) score: int
            checked_players[player_thieu] = score

    if best_match:
        matches.append(best_match)
        matched_players.add(player_thieu)
        if player_thieu in checked_players:
            del checked_players[player_thieu]

    if (index + 1) % 50 == 0 or (index + 1) == len(df_thieu):
        print(f"✅ Processed {index + 1}/{len(df_thieu)} rows in Thieu.csv...")
```

- All matched players with their similarity scores are saved to ETV2.csv.

```python
# Save matches to ETV2.csv
matches_df = pd.DataFrame(matches, columns=['Player', 'Squad', 'Player_Thieu', 'Squad_Thieu', 'Similarity'])
matches_df.to_csv('ETV2.csv', index=False)

print(f"\n✅ Matched successfully: {len(matches_df)} players.")
print(f"⚠ Players checked but not matched: {len(checked_players)}")
```

- Merges the matches in ETV2.csv with GiaTriCauThu.csv to retrieve their ETV values.

```python
# Merge with GiaTriCauThu.csv to get ETV
merged_etv2 = pd.merge(
    matches_df,
    df_gt[['Player', 'Squad', 'ETV']],
    on=['Player', 'Squad'],
    how='left'
)
```

- Combines the merged data with the original ETV.csv, ensuring no duplicates are present and sorting the data by player name.

```python
# Combine with original ETV.csv to avoid duplicates
combined_etv = pd.concat([df_etv[['Player', 'Squad', 'ETV']], df_etv2_final], ignore_index=True)
combined_etv.drop_duplicates(subset=['Player', 'Squad'], keep='first', inplace=True)
```

- The ﬁnal result is saved back to ETV.csv with a new index

```python
# Sort and reassign index
combined_etv = combined_etv.sort_values(by='Player').reset_index(drop=True)
combined_etv.insert(0, 'STT', range(1, len(combined_etv) + 1))

# Save final result
combined_etv.to_csv('ETV.csv', index=False, encoding='utf-8-sig')

print(f"\n📦 Total rows in final ETV.csv: {len(combined_etv)}")
```

1.2.8 Output

- ETV2.csv: Contains matched players from Thieu.csv and CheckLan2.csv with their similarity scores.

```
Player,Squad,Player_Thieu,Squad_Thieu,Similarity
Alphonse Aréola,West Ham,Alphonse Areola,West Ham,93
Arijanet Murić,Ipswich Town,Arijanet Muric,Ipswich Town,93
Idrissa Gueye,Everton,Idrissa Gana Gueye,Everton,69
Simon Adingra,Brighton,Igor,Brighton,75
Ismaïla Sarr,Crystal Palace,Ismaila Sarr,Crystal Palace,92
Jérémy Doku,Manchester City,Jeremy Doku,Manchester City,82
Jurrien Timber,Arsenal,Jurriën Timber,Arsenal,93
Manuel Ugarte,Manchester Utd,Manuel Ugarte Ribeiro,Manchester Utd,100
Pedro Lima,Wolves,Mario Lemina,Wolves,60
Miloš Kerkez,Bournemouth,Milos Kerkez,Bournemouth,92
O. Hutchinson,Ipswich Town,Omari Hutchinson,Ipswich Town,85
Radu Drăgușin,Tottenham,Radu Drăgușin,Tottenham,92
Rasmus Winther Højlund,Manchester Utd,Rasmus Højlund,Manchester Utd,64
Rayan Aït Nouri,Wolves,Rayan Aït-Nouri,Wolves,93
Heung-min Son,Tottenham,Son Heung-min,Tottenham,69
Victor Kristiansen,Leicester City,Victor Bernth Kristiansen,Leicester City,72
```

- ETV.csv: Final list of players (with matches from both ETV.csv and ETV2.csv), sorted and without duplicates, including their Estimated Transfer Values (ETVs).

```
STT,Player,Squad,ETV
253,Sam Morsy,Ipswich Town,€1.3M
254,Sammie Szmodics,Ipswich Town,€14.1M
255,Sander Berge,Fulham,€27.8M
256,Sandro Tonali,Newcastle Utd,€52.9M
257,Santiago Bueno,Wolves,€11.2M
258,Saša Lukić,Fulham,€12.1M
259,Sepp van den Berg,Brentford,€26.4M
260,Simon Adingra,Brighton,€34M
261,Stefan Ortega,Manchester City,€5.8M
262,Stephy Mavididi,Leicester City,€12.9M
263,Sávio,Manchester City,€74.8M
264,Taylor Harwood-Bellis,Southampton,€29.9M
265,Thomas Partey,Arsenal,€15.2M
266,Timothy Castagne,Fulham,€17.4M
267,Tomáš Souček,West Ham,€23.2M
268,Tosin Adarabioyo,Chelsea,€24.3M
269,Toti Gomes,Wolves,€18.6M
270,Trent Alexander-Arnold,Liverpool,€55.8M
271,Trevoh Chalobah,Crystal Palace,€20.1M
272,Tyler Adams,Bournemouth,€24.4M
273,Tyler Dibling,Southampton,€32.7M
274,Tyrick Mitchell,Crystal Palace,€24.6M
275,Tyrone Mings,Aston Villa,€5.8M
276,Valentino Livramento,Newcastle Utd,€49.3M
277,Victor Kristiansen,Leicester City,€23.9M
278,Virgil van Dijk,Liverpool,€24.3M
279,Vitaliy Mykolenko,Everton,€33.5M
280,Vitaly Janelt,Brentford,€27.1M
281,Wes Burns,Ipswich Town,€2.1M
282,Wesley Fofana,Chelsea,€30.3M
283,Wilfred Ndidi,Leicester City,€20.3M
284,Will Hughes,Crystal Palace,€11.1M
285,William Saliba,Arsenal,€79.5M
286,Wout Faes,Leicester City,€25M
287,Yankuba Minteh,Brighton,€42.6M
288,Yasin Ayari,Brighton,€21.4M
289,Yehor Yarmoliuk,Brentford,€18M
290,Yoane Wissa,Brentford,€31.4M
291,Youri Tielemans,Aston Villa,€41.5M
292,Yukinari Sugawara,Southampton,€15.7M
293,Yves Bissouma,Tottenham,€29.8M
294,Álex Moreno,Nott'ham Forest,€11M
295,İlkay Gündoğan,Manchester City,€12.1M
296,Łukasz Fabiański,West Ham,€0.9M
```

```
(venv) PS C:\Users\Nguyen Duc Tin\OneDrive\Desktop\BTLPython\Bai4\Phan1> python ETV2.py
🔍 Total rows in CheckLan2.csv: 253
📋 Total rows in Thieu.csv: 19
✅ Processed 19/19 rows in Thieu.csv...

✅ Matched successfully: 16 players.
⚠️Players checked but not matched: 3

📦 Total rows in final ETV.csv: 296
```

- Result: This method only works 16/19 with a rate of 84.21%
    - with 16 players successfully matched, including 2 players who matched incorrectly and 3 players have transferred so there is no data: Adam Armstrong, Mads Roerslev, Kyle Walker

2. Propose a method for estimating player values.

2.1 Make data for training

2.1.1 Objective Overview

- This script is designed to generate three key evaluation scores for football players age score, playing time score, and performance score based on match statistics and demographic information. The resulting data is saved to a ﬁle DataDanhGia.csv for downstream analysis or model training.

2.1.2 Technology Description

- Pandas is used for data loading, transformation, and export.
- Scikit-learn s MinMaxScaler is applied to normalize numerical performance features to a 0 1 range.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

2.1.3 Code Breakdown

- TimeScore(minutes): Converts total minutes played into a score to reﬂect player involvement in matches. The more minutes, the higher the score, with a maximum score of 1.0 for ≥2500 minutes.

```
# Function to calculate score based on playing time
def TimeScore(minutes):
    if pd.isna(minutes):
        return 0.0
    try:
        minutes = float(minutes)
    except ValueError:
        return 0.0
    if minutes < 900: return 0.2
    elif minutes < 1500: return 0.4
    elif minutes < 2000: return 0.6
    elif minutes < 2500: return 0.8
    else: return 1.0
```

- PerformanceScore(row): Calculates a weighted performance score based on the player s position and their relevant metrics. The weights vary depending on whether the player is a goalkeeper (GK), defender (DF), mid%elder (MF), or forward (FW).

If a player has a hybrid position like DF-MF, the total score is divided accordingly.

```python
# Function to calculate performance score
def PerformanceScore(row):
    pos = row['Pos']
    score = 0
    if not isinstance(pos, str) or pos.strip() == '':
        return 0.0
    if 'GK' in pos:
        score += row.get('GA90', 0) * 0.5
        score += row.get('Save%', 0) * 0.6
        score += row.get('CS%', 0) * 0.4
        score += row.get('Save%.1', 0) * 0.3
    if 'DF' in pos:
        score += row.get('Tkl', 0) * 0.5
        score += row.get('TklW', 0) * 0.4
        score += row.get('Attd', 0) * 0.3
        score += row.get('Lostd', 0) * -0.2
        score += row.get('Blocks', 0) * 0.5
        score += row.get('Sh', 0) * 0.3
        score += row.get('Pass', 0) * 0.2
        score += row.get('Int', 0) * 0.4
    if 'MF' in pos:
        score += row.get('Cmp', 0) * 0.4
        score += row.get('Cmp%', 0) * 0.3
        score += row.get('TotDist', 0) * 0.2
        score += row.get('Cmp%.1', 0) * 0.3
        score += row.get('Cmp%.2', 0) * 0.2
        score += row.get('Cmp%.3', 0) * 0.2
        score += row.get('KP', 0) * 0.4
        score += row.get('Pto1/3', 0) * 0.3
        score += row.get('PPA', 0) * 0.3
        score += row.get('CrsPA', 0) * 0.2
        score += row.get('PrgPp', 0) * 0.5
    if 'FW' in pos:
        score += row.get('Gls', 0) * 0.6
        score += row.get('Ast', 0) * 0.5
        score += row.get('CrdY', 0) * -0.1
        score += row.get('CrdR', 0) * -0.2
        score += row.get('xG', 0) * 0.7
        score += row.get('xAG', 0) * 0.6
        score += row.get('PrgCs', 0) * 0.3
        score += row.get('PrgPs', 0) * 0.3
        score += row.get('PrgRs', 0) * 0.3
    return round(score / len(pos.split('-')) if '-' in pos else score, 2)
```

- AgeScore(age): Converts a player's age into a normalized score, where younger players receive higher values. Special cases such as NaN or age in string format (e.g., "21-2") are handled gracefully.
  - 18 → 1.0
  - 19 21 → 0.9
  - 22 24 → 0.8
  - 25 27 → 0.6
  - 28 30 → 0.4
  - 31 33 → 0.2
  - 33 → 0.1

```python
# Function to calculate score based on age
def AgeScore(age):
    if pd.isna(age):
        return 0.0
    if isinstance(age, str) and '-' in age:
        age = age.split('-')[0]
    try:
        age = int(age)
    except ValueError:
        return 0.0
    if age <= 18: return 1.0
    elif age <= 21: return 0.9
    elif age <= 24: return 0.8
    elif age <= 27: return 0.6
    elif age <= 30: return 0.4
    elif age <= 33: return 0.2
    else: return 0.1
```

- NormalizePercentage(df, columns): Takes a list of percentage-based column names and converts each to a decimal value (e.g., 75% → 0.75).Columns handled: ['Save%', 'SoT%', 'Cmp%', 'CS%', 'Won%', 'Save%.1'].
- NormalizeMinMax(df, columns): Applies Min-Max scaling to ensure all values fall within the [0, 1] range. Only numeric and existing columns are processed to avoid errors. The selected columns include various statistics like passes, tackles, goals, and expected goals.

- The dataset is read from ChoiTren900p.csv.

```
# Load data
df = pd.read_csv('ChoiTren900p.csv')
```

- Percentage and performance columns are normalized using the two functions above.

```
# Normalize percentage columns
columns_percent = ['Save%', 'SoT%', 'Cmp%', 'CS%', 'Won%', 'Save%.1']
df = NormalizePercentage(df, columns_percent)
```

- NaN values in relevant columns are replaced with 0 to ensure smooth computation during score calculation

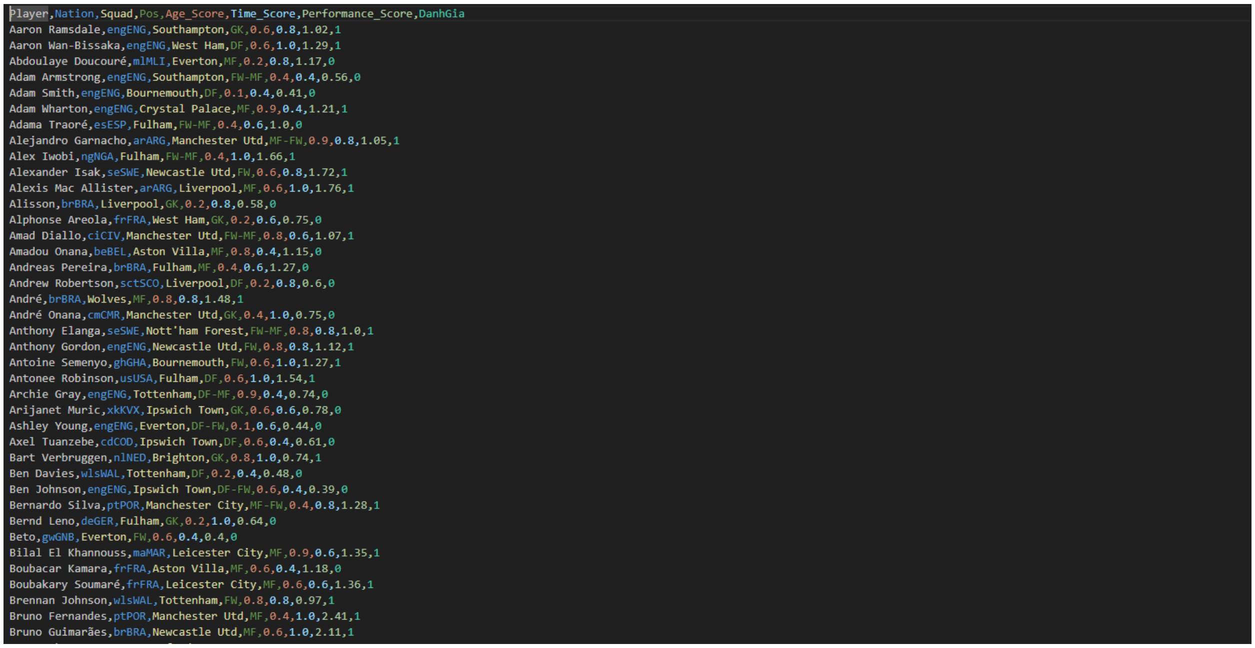
```
# Fill NaN with 0 in selected columns
for col in columns_minmax + columns_percent:
    if col in df.columns:
        df[col] = df[col].fillna(0)
```

- Calculated Age_Score, Time_Score, Performance_Score, DanhGia

```
# Calculate auxiliary scores
df['Age_Score'] = df['Age'].apply(AgeScore)
df['Time_Score'] = df['Min'].apply(TimeScore)

# Calculate performance score
df['Performance_Score'] = df.apply(PerformanceScore, axis=1)

# Calculate overall rating (DanhGia) based on a threshold (e.g., 0.5)
df['DanhGia'] = (df['Performance_Score'] + df['Age_Score'] + df['Time_Score']) / 3
df['DanhGia'] = df['DanhGia'].apply(lambda x: 1 if x >= 0.8 else 0)
```

2.1.4 Output

- This final dataset is saved as DataDanhGia.csv for further use in analysis or modeling.



```
Player,Nation,Squad,Pos,Age_Score,Time_Score,Performance_Score,DanhGia
Aaron Ramsdale,engENG,Southampton,GK,0.6,0.8,1.02,1
Aaron Wan-Bissaka,engENG,West Ham,DF,0.6,1.0,1.29,1
Abdoulaye Doucouré,mlMLI,Everton,MF,0.2,0.8,1.17,0
Adam Armstrong,engENG,Southampton,FW-MF,0.4,0.4,0.56,0
Adam Smith,engENG,Bournemouth,DF,0.1,0.4,0.41,0
Adam Wharton,engENG,Crystal Palace,MF,0.9,0.4,1.21,1
Adama Traoré,esESP,Fulham,FW-MF,0.4,0.6,1.0,0
Alejandro Garnacho,arARG,Manchester Utd,MF-FW,0.9,0.8,1.05,1
Alex Iwobi,ngNGA,Fulham,FW-MF,0.4,1.0,1.66,1
Alexander Isak,seSWE,Newcastle Utd,FW,0.6,0.8,1.72,1
Alexis Mac Allister,arARG,Liverpool,MF,0.6,1.0,1.76,1
Alisson,brBRA,Liverpool,GK,0.2,0.8,0.58,0
Alphonse Areola,frFRA,West Ham,GK,0.2,0.6,0.75,0
Amad Diallo,ciCIV,Manchester Utd,FW-MF,0.8,0.6,1.07,1
Amadou Onana,beBEL,Aston Villa,MF,0.8,0.4,1.15,0
Andreas Pereira,brBRA,Fulham,MF,0.4,0.6,1.27,0
Andrew Robertson,sctSCO,Liverpool,DF,0.2,0.8,0.6,0
André,brBRA,Wolves,MF,0.8,0.8,1.48,1
André Onana,cmCMR,Manchester Utd,GK,0.4,1.0,0.75,0
Anthony Elanga,seSWE,Nott'ham Forest,FW-MF,0.8,0.8,1.0,1
Anthony Gordon,engENG,Newcastle Utd,FW,0.8,0.8,1.12,1
Antoine Semenyo,ghGHA,Bournemouth,FW,0.6,1.0,1.27,1
Antonee Robinson,usUSA,Fulham,DF,0.6,1.0,1.54,1
Archie Gray,engENG,Tottenham,DF-MF,0.9,0.4,0.74,0
Arijanet Muric,xkKVX,Ipswich Town,GK,0.6,0.6,0.78,0
Ashley Young,engENG,Everton,DF-FW,0.1,0.6,0.44,0
Axel Tuanzebe,cdCOD,Ipswich Town,DF,0.6,0.4,0.61,0
Bart Verbruggen,nlNED,Brighton,GK,0.8,1.0,0.74,1
Ben Davies,wlsWAL,Tottenham,DF,0.2,0.4,0.48,0
Ben Johnson,engENG,Ipswich Town,DF-FW,0.6,0.4,0.39,0
Bernardo Silva,ptPOR,Manchester City,MF-FW,0.4,0.8,1.28,1
Bernd Leno,deGER,Fulham,GK,0.2,1.0,0.64,0
Beto,gwGNB,Everton,FW,0.6,0.4,0.4,0
Bilal El Khannouss,maMAR,Leicester City,MF,0.9,0.6,1.35,1
Boubacar Kamara,frFRA,Aston Villa,MF,0.6,0.4,1.18,0
Boubakary Soumaré,frFRA,Leicester City,MF,0.6,0.6,1.36,1
Brennan Johnson,wlsWAL,Tottenham,FW,0.8,0.8,0.97,1
Bruno Fernandes,ptPOR,Manchester Utd,MF,0.4,1.0,2.41,1
Bruno Guimarães,brBRA,Newcastle Utd,MF,0.6,1.0,2.11,1
```

2.2 Training model (TrainModel.py)

2.2.1 Objective Overview

- This module aims to build a predictive model that estimates a football player's transfer value trend (DanhGia) based on three key evaluation criteria: age-related performance (Age_Score), playing time (Time_Score), and professional performance (ChuyenMon_Score). The goal is to provide an automated, data-driven approach to assist in player valuation by learning patterns from historical player data.

2.2.2 Technology Description

- Pandas: Used for reading the dataset from a CSV file and handling structured tabular data
- Scikit-learn (train_test_split): Splits the dataset into training, validation, and testing subsets to ensure fair model evaluation.
- Keras (with TensorFlow backend):
  - Sequential model: A straightforward neural network container for layer-by-layer model building.

- Dense layer: Implements a fully connected linear output layer with a single neuron for regression output.
- Loss Function: mean_squared_error is chosen to penalize large prediction errors.
- Optimizer: adam optimizer for e cient gradient-based training.
- Metrics: Uses Mean Absolute Error (MAE) to monitor model performance during training.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
```

2.2.3 Code Breakdown

- The CSV ﬁle DataDanhGia.csv is read using pandas.

```
# Read the CSV file
df = pd.read_csv('DataDanhGia.csv', skipinitialspace=True)
```

- X: Uses three input features Age_Score, Time_Score, and ChuyenMon_Score for training.
- y: Target variable DanhGia, the evaluation score representing the player's value change.

```
X = df.iloc[:, 4:7].values  # Use Age_Score, Time_Score, ChuyenMon_Score
y = df.iloc[:, 7].values    # Use DanhGia (evaluation score)
```

- The dataset is split into: Training + Validation set (85%) and Test set (15%)
- Then, from the 85%, another split creates: Training set (~70%) and Validation set (~15%)

  ⇨ This results in a 70/15/15 split across train/validation/test.
- A simple linear regression model using Keras Sequential API is built.
- The network has:
  - One input layer with 3 neurons (matching 3 input features).
  - One output layer with 1 neuron for predicting the evaluation score.

- The model is compiled with:
  - Loss function: Mean Squared Error (MSE) standard for regression tasks.
  - Optimizer: Adam adaptive learning rate optimizer.
  - Evaluation metric: Mean Absolute Error (MAE).
- The model is trained on the training set with:
  - 100 epochs
  - Batch size of 10
  - Validation data is used during training to monitor performance.

2.2.4 Output

- Model file: GiaTriChuyenNhuong.h5
  This file contains the weights and architecture of the trained regression model and can be reused for predictions without retraining.

- Metrics of the model when tested: (TestModel.py)
  - Predict on the first 44 test samples

```
📈 Mean Absolute Error (MAE): 0.2000
📉 Root Mean Squared Error (RMSE): 0.4472
📊 R² Score: 0.1667
✅ Accuracy: 0.8000
🎯 Precision: 0.8000
🔁 Recall: 0.6667
```

```
Comparison of actual vs predicted (first 20 samples):
    Actual  Predicted   Score
0        1          1  0.6412
1        1          0  0.4924
2        1          0  0.3021
3        1          1  0.8837
4        0          0  0.4408
5        1          0  0.4964
6        0          1  0.6861
7        1          1  0.6766
8        1          0  0.3373
9        0          0  0.2517
10       1          0  0.3916
11       0          0  0.2461
12       0          0 -0.0102
13       0          0  0.3210
14       0          0 -0.0057
15       0          0  0.3167
16       1          1  0.9131
17       0          0  0.3201
18       1          1  0.9782
19       0          0  0.0602
```

2.2.5 Answer the question

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

- Proposed Method for Estimating Player Values

1. Feature Selection:
    - Use key metrics like Age_Score, Time_Score, and ChuyenMon_Score (skills, age, experience).
    - Add features like Goals, Assists, Minutes Played, and Pass Completion for more insight.
2. Model Selection:
    - Start with Linear Regression (simple, interpretable).
    - Use more complex models like Random Forest or Gradient Boosting if needed for non-linear patterns.
3. Preprocessing:
    - Normalize features and handle missing values to improve model accuracy.

4. Training and Evaluation:
   - Split data into Train, Validation, and Test sets.
   - Evaluate using MAE, RMSE, and R^2.
5. Deployment
   - After training, use the model to predict player values and analyze feature importance.