**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**

**Table of Contents…………………………………………………………………………………..2**

**Problem I……………………………………………………………………………………………..3**

**1. Claw Data (LayData.py)………………………………………………………………………3**

**2. Merge Data (EPLPlayer.py)………………………………………………………………….4**

**Problem II…………………………………………………………………………………………..12**

**1. Top 3 players with the highest and lowest scores (Top3Player.py)………………..12**

**2. Median of data (TrungVi.py)……………………………………………………………….15**

**3.** **A histogram plots the distribution of each statistic for all players in the league and each team (Ve.py & VeTeam.py)….……………………………………………………18**

**4. Best Performing Team in the 2024-2025 Premier League season (DoiThanhTichTotNhat.py)……………………………………………………………26**

**5. Representative Offensive and Defensive Statistics(ATK\_DF.py)………...28**

**Problem III………………………………………………………………………………………….31**

1. **K-means algorithm (TimK.py)………………………………………….…..…….31**
2. **Plot a 2D cluster of the data points (PhanLoaiCauThu.py)……………....36**

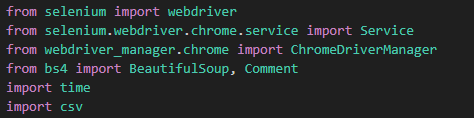
**Problem IV……………………………………………………………………………………….…38**

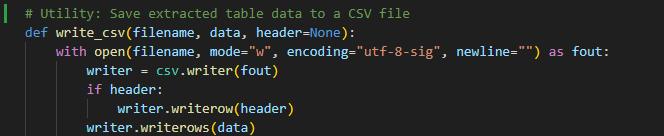
1. **Collect the transfer values of players for the 2024-2025 season from** [**https://www.footballtransfers.com/**](https://www.footballtransfers.com/) **whose playing time exceeds 900 minutes………………………………………………………………………………….38**
2. **Propose a method for estimating player values………………………………..51**

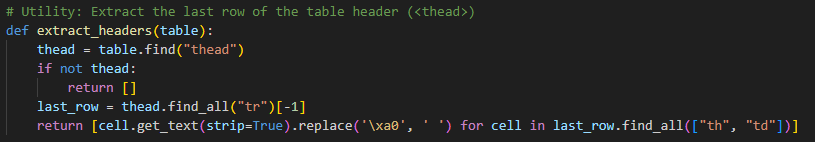
**Problem I**

1. **Claw Data (LayData.py)**
   1. **Objective Overview**

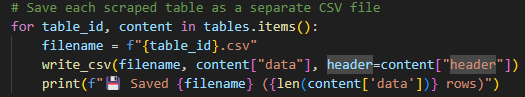
* **Automate the extraction of player statistics from FBref.com for the 2024–2025 Premier League season, specifically targeting various performance categories like standard stats, shooting, passing, goalkeeping, defense, etc. Each category will be saved in a separate CSV file.**
  1. **Technology Description**
* **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**

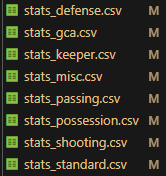
****

* 1. **Code Breakdown**
* **write\_csv(): Saves extracted data into CSV with or without a header.** ****
* **extract\_headers(): Extracts table headers from the last row of <thead>.**

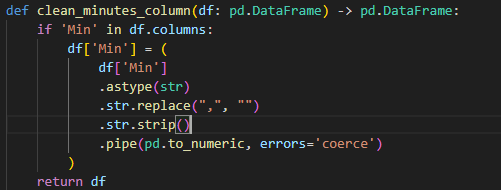
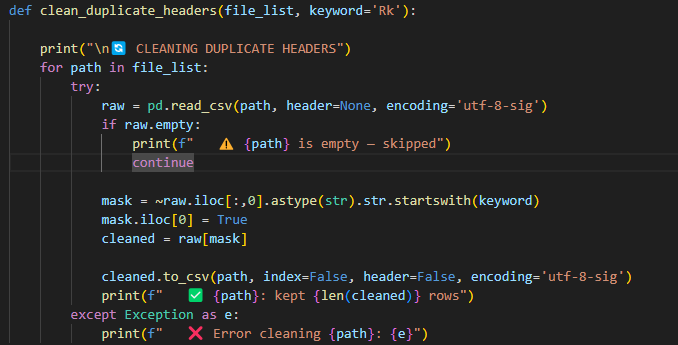
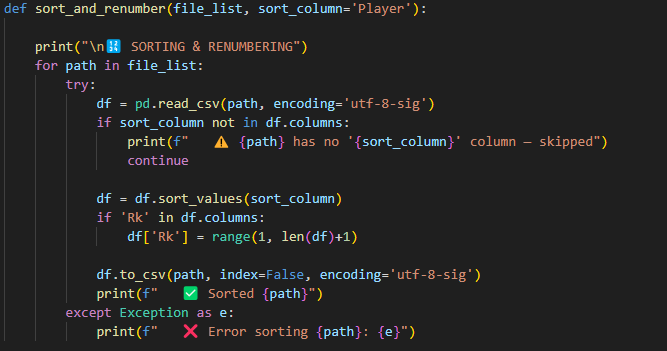
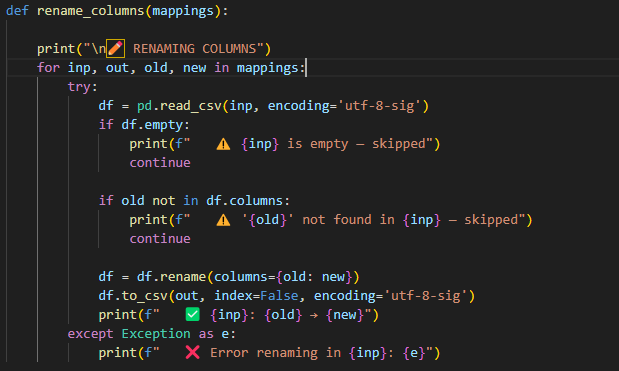
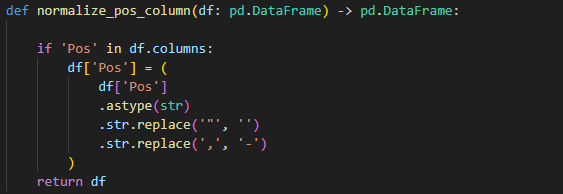
****

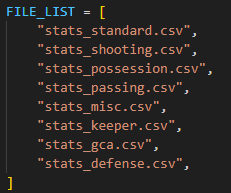
* **Web Scraping:**
* **Uses Selenium to load pages, waits for content to load.**
* **Parses HTML with BeautifulSoup, extracts tables by their IDs.**
* **Data Processing: Cleans table data, ensuring consistency**
* **Saving Data: Saves each table's data in a separate CSV file**

****

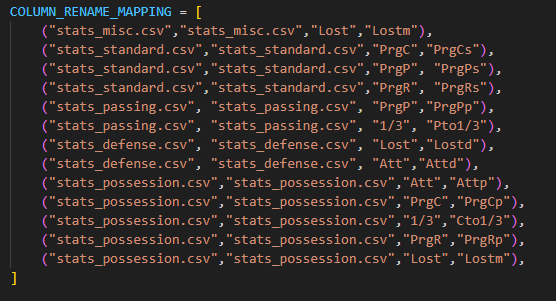
* 1. **Output**
* **Generates CSV files for each category (e.g., stats\_standard.csv, stats\_shooting.csv) containing player statistics.** ****

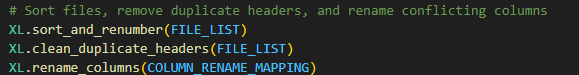
1. **Merge Data (EPLPlayer.py)**
   1. **Objective Overview**

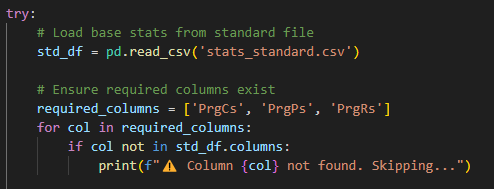
* **Merge multiple CSV files containing different categories of football player statistics (e.g., shooting, passing, possession, defense) into a unified dataset.**
* **Ensure data consistency, avoid column name collisions, and filter for players who have played more than 90 minutes in total.**
* **Export the final cleaned and merged dataset to results.csv for further analysis or modeling.**
  1. **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** ****
* **def clean\_duplicate\_header(): Remove duplicate header lines in the file caused by crawling from FBref (usually the first line is "Rk").** ****
* **def sort\_and\_renumber(): Sort players by first name (Player) and renumber the Rk column if it exists.** ****
* **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.** ****
* **def normalize\_pos\_column(): Normalize “Pos” column for later processing.** ****
  1. **Code Breakdown**
* **FILE\_LIST: Lists all raw CSVs that contain the statistics to be merged.**

****

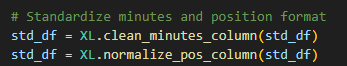
* **COLUMN\_RENAME\_MAPPING: Handles potential column name conflicts across different files by mapping old names to new, unique ones.**

****

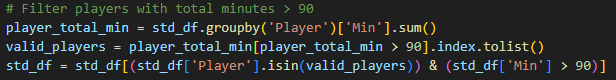
* **HeaderGroups & FileMap: Define which columns to keep from each file and the corresponding file names.** ****
* **Preprocessing:**
* **sort\_and\_renumber(): Sorts files and resets headers if necessary.**
* **clean\_duplicate\_headers(): Removes redundant headers that may exist within data rows.**
* **rename\_columns(): Renames conflicting column names as defined in the mapping.** ****
* **Base Dataset Creation:**
* **Loads stats\_standard.csv**

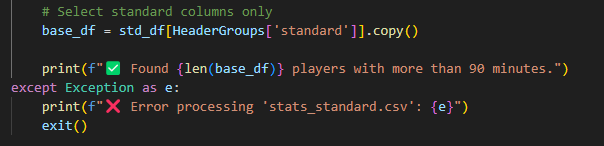
****

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

****

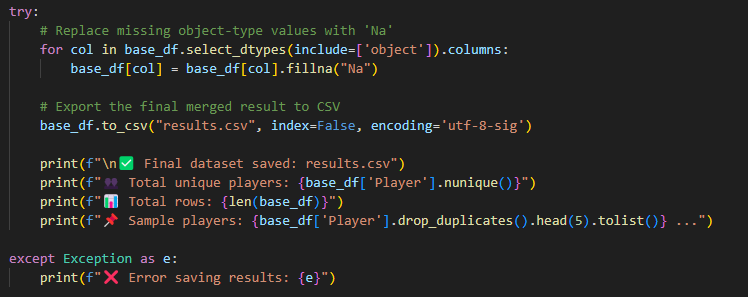
* **Filters out players with total minutes played ≤ 90.**

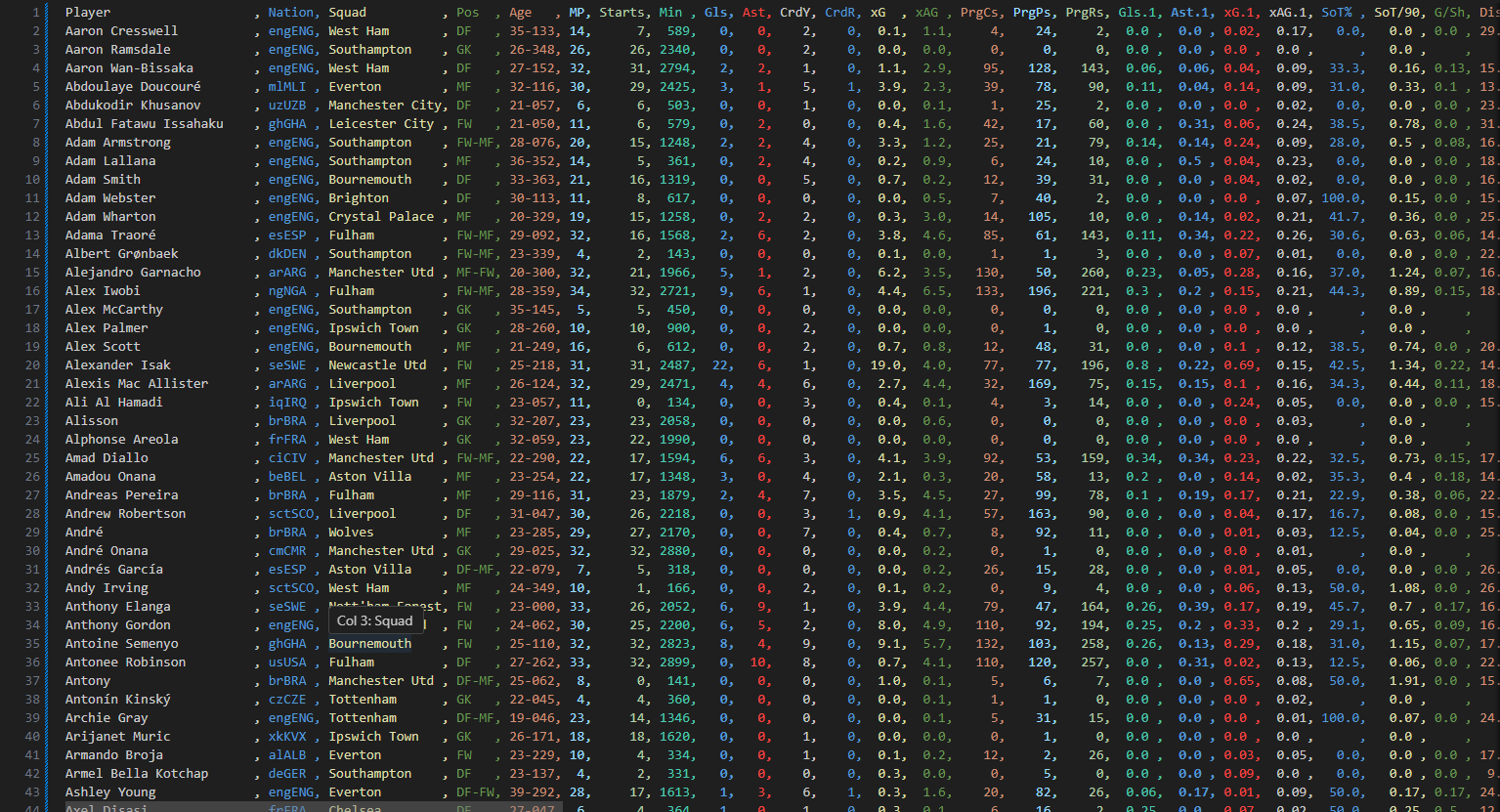
****

* **Selects and saves only the required columns defined under standard group.** ****
* **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 file not found or failing to process.**

****

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

****

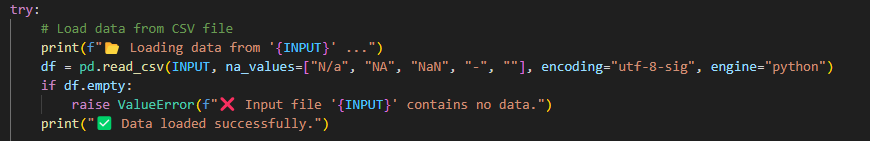
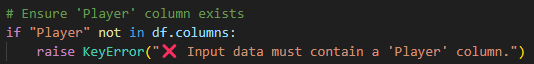
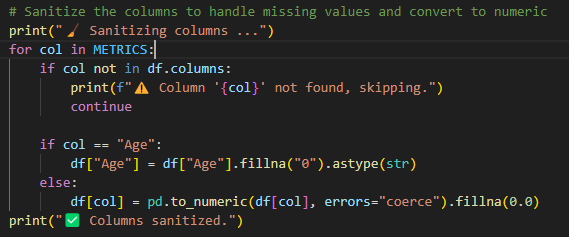
* 1. **Output**
* **A unified results.csv file containing cleaned, de-duplicated, and merged player statistics across all performance categories.**
* **Ensures only players with significant play time (over 90 minutes) are included.**
* **Data is ready for use in downstream tasks such as modeling, analysis, or visualization.** ****

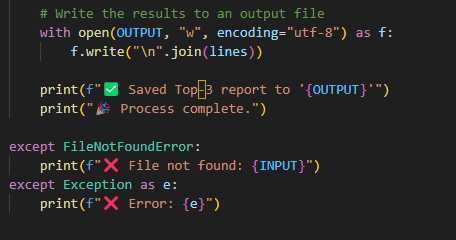
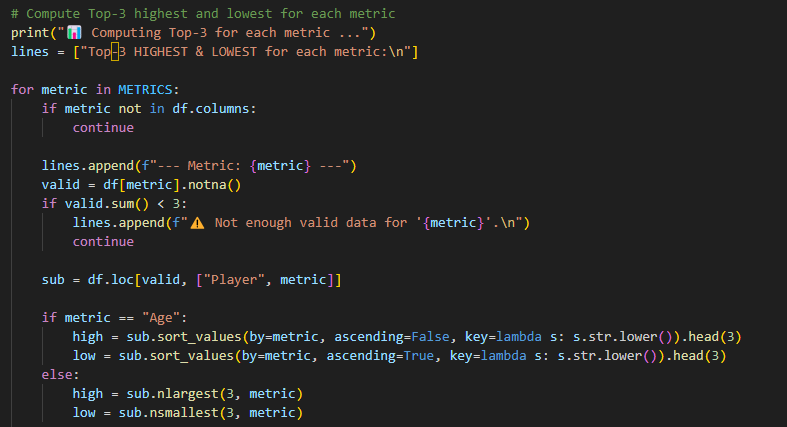
**Problem II**

1. **Top 3 players with the highest and lowest scores (Top3Player.py)**
   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. **Technology Description**
* **Pandas: Handles CSV reading, numeric conversion, and ranking operations efficiently.**

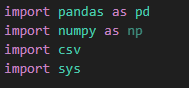
****

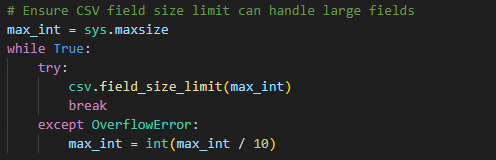
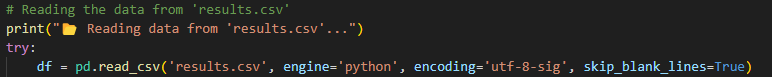
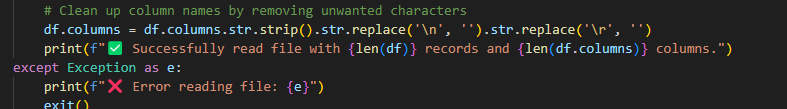
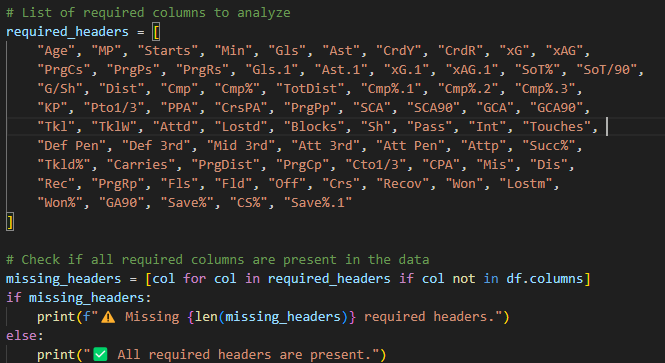
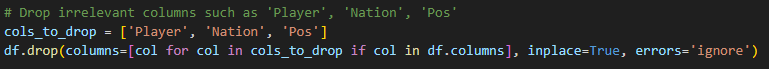
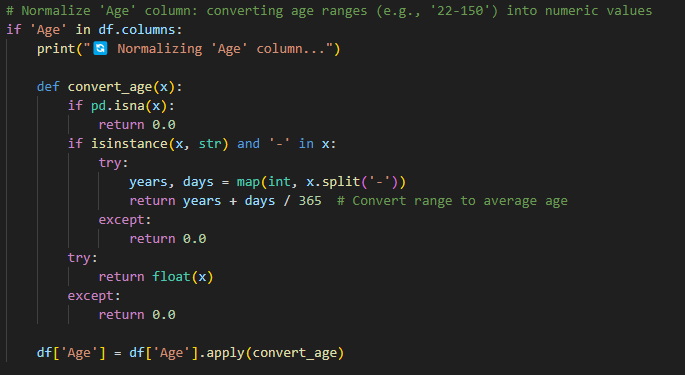
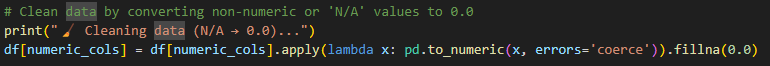
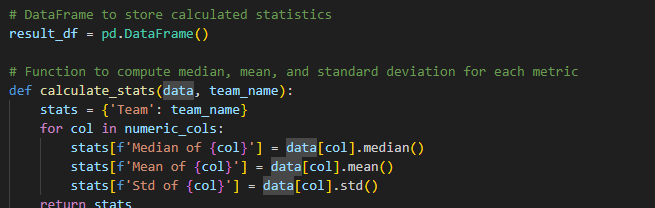
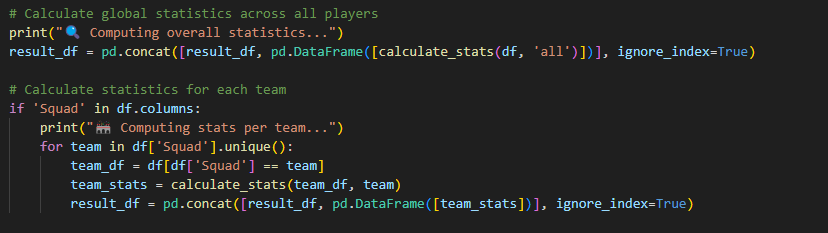
* **Python Built-in File I/O: Used to write the Top‑3 results to a text file in structured format.**
  1. **Code Breakdown**
* **METRICS: A curated list of numeric or string-based performance metrics to be evaluated.**
* **INPUT / OUTPUT: File paths for the input data (results.csv) and output report (top3.txt).**
* **Loading and Validation:**
* **Reads results.csv using UTF-8-SIG encoding.** ****
* **Validates that the file contains data and that required columns (e.g., Player) exist.** ****
* **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.** ****

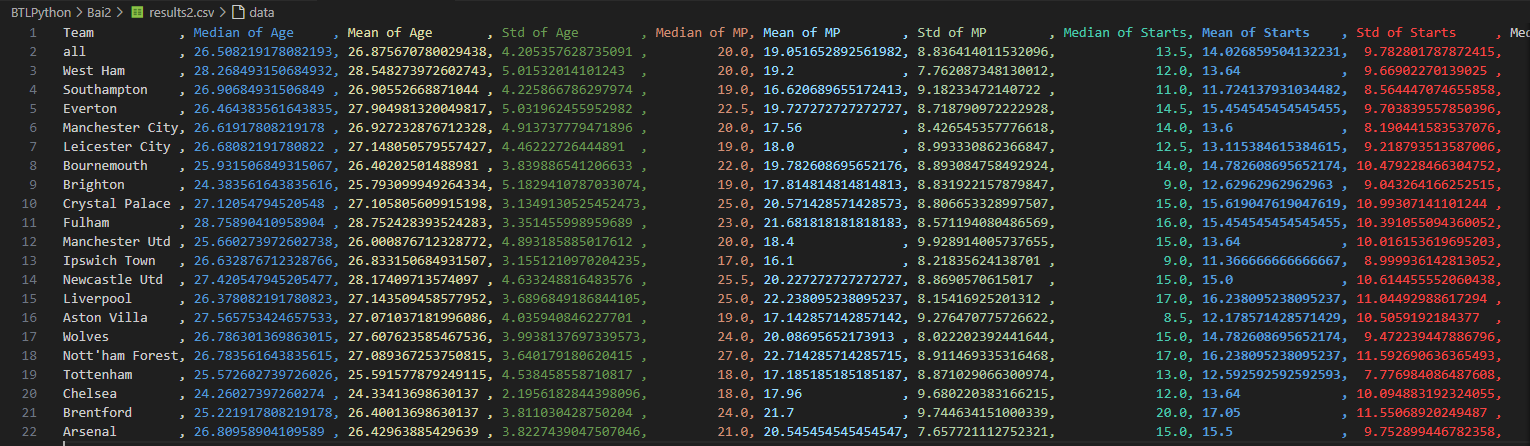
* **Top‑3 Computation:**
* **Filters valid rows**
* **Computes Top‑3 highest and lowest values using nlargest() and nsmallest() (or string sort for Age)**
* **Formats results and adds them to a text buffer**
* **Output Writing:**
* **Writes all metric summaries into a clean, structured file (top3.txt).** ****
* **Each section includes the metric name, Top‑3 highest and lowest performers, and separators for readability.** ****
  1. **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.** ****

1. **Median of data (TrungVi.py)**
   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**
  1. **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.**

****

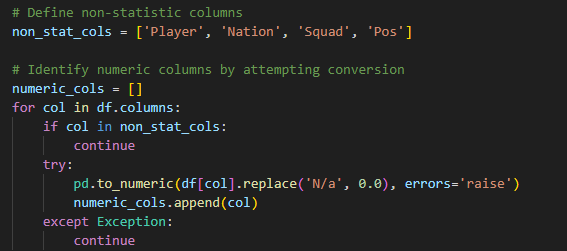
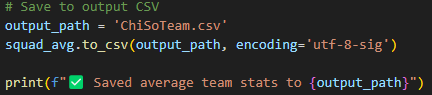
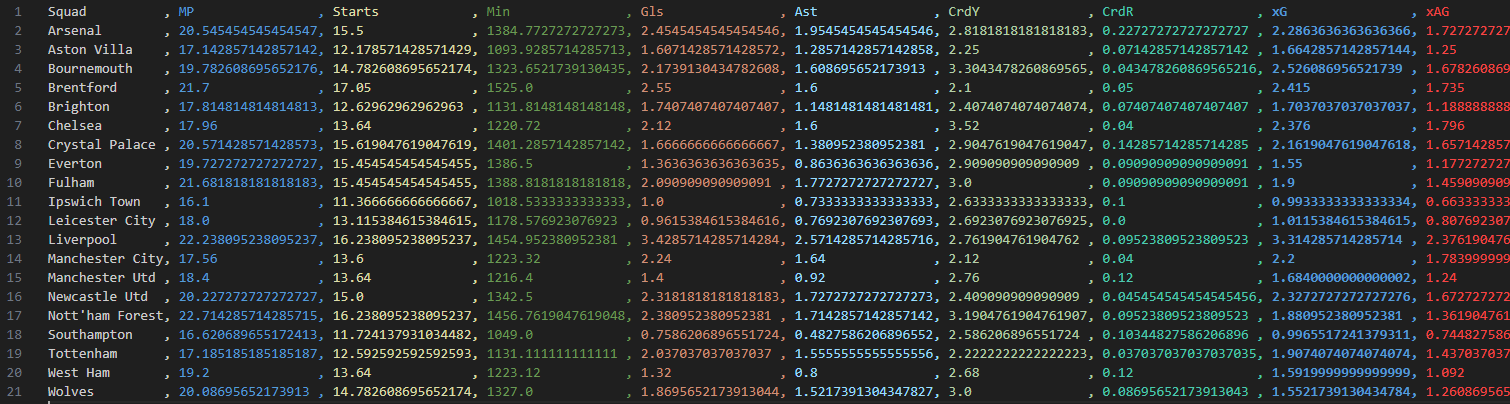
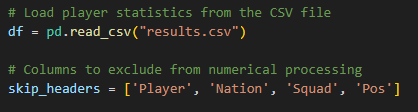
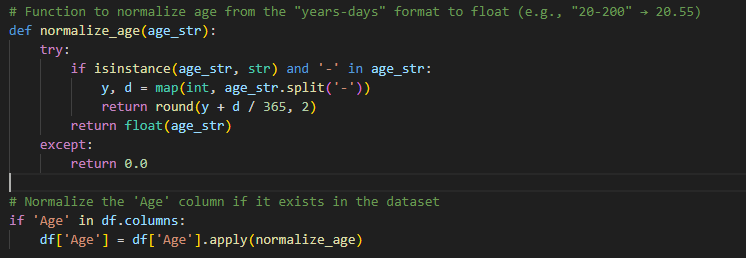
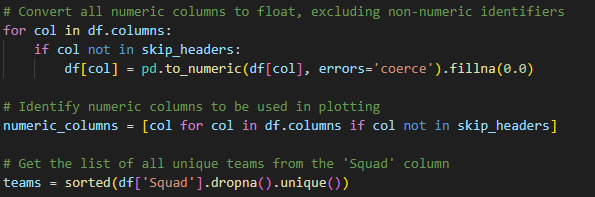
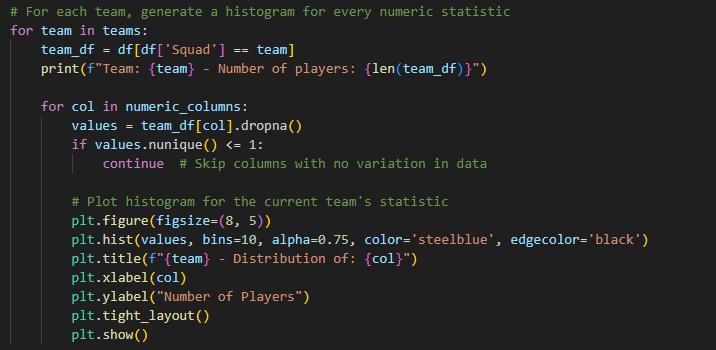
* 1. **Code Breakdown**
* **Sets the CSV field size limit using sys.maxsize to accommodate large datasets.** ****
* **Reads the CSV file using pandas.read\_csv with UTF-8 encoding.** ****
* **Cleans column names by removing newline and carriage return characters.** ****
* **Validates whether all necessary performance metrics (like Gls, xAG, Touches, etc.) exist in the dataset.** ****
* **Drops irrelevant fields such as Player, Nation, and Pos.** ****
* **Normalizes the Age column by converting values like '22-150' to a decimal form (22 + 150/365).** ****
* **Ensures all numeric columns are properly converted.** ****
* **Replaces non-numeric and missing values ('N/A', empty cells, etc.) with 0.0.** ****
* **For each performance metric: median, mean, standard deviation**
* **Once across all players.** ****
* **Separately for each team (using the Squad column).** ****
* **Results are written to results2.csv.** ****
* **The final file includes:**
* **One row for overall statistics (Team = all)**
* **One row for each team**
  1. **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>)**

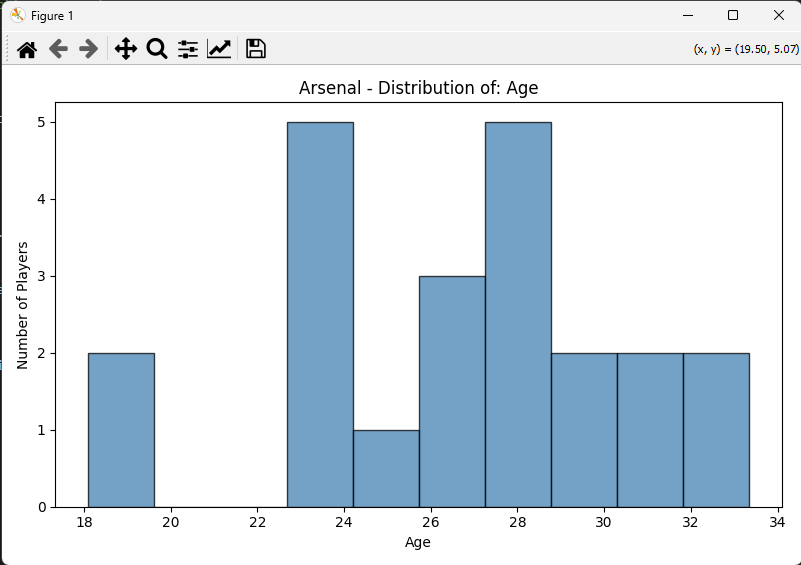
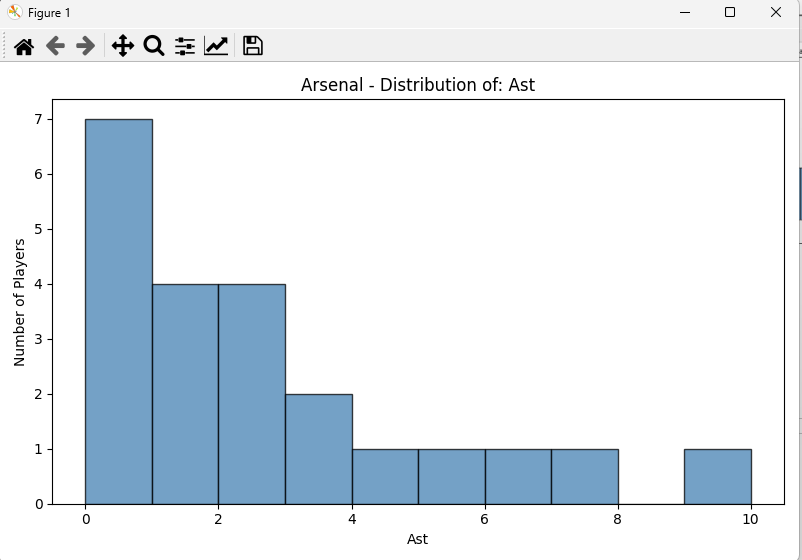
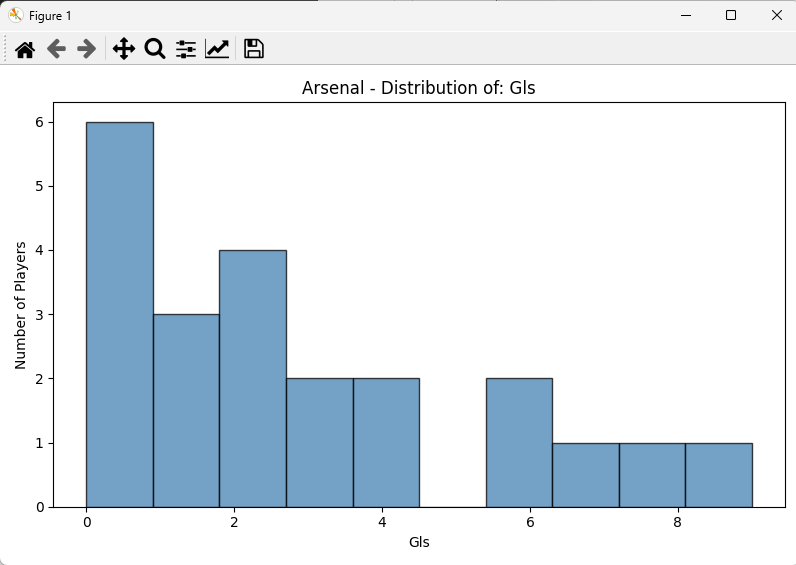
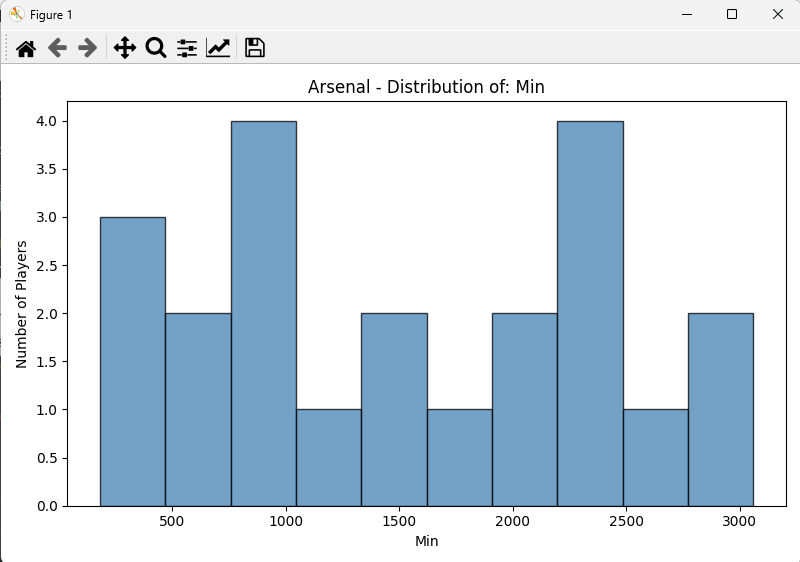
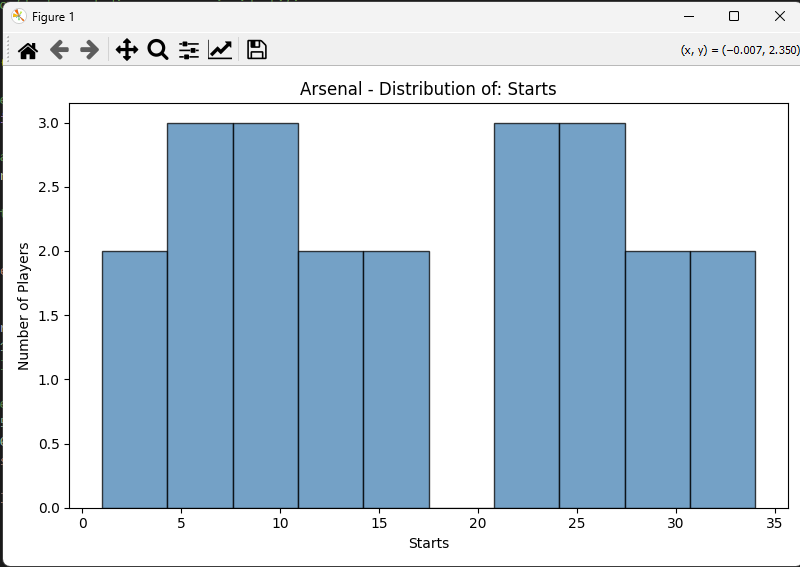
****

1. **A histogram plots the distribution of each statistic for all players in the league and each team.**
   1. **For each team (VeTeam.py)**
      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).**

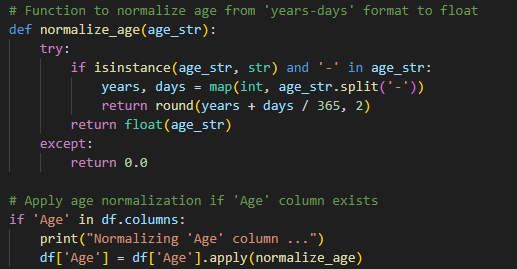
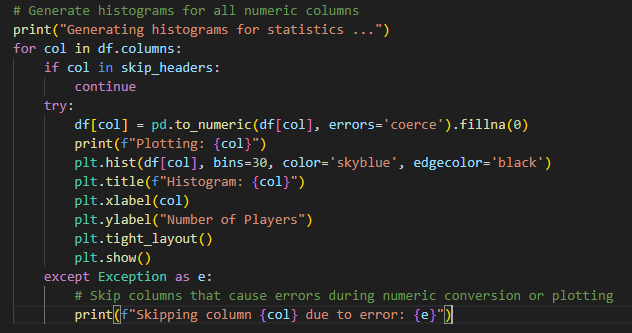
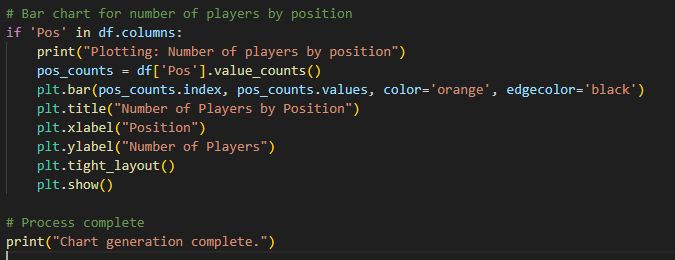
****

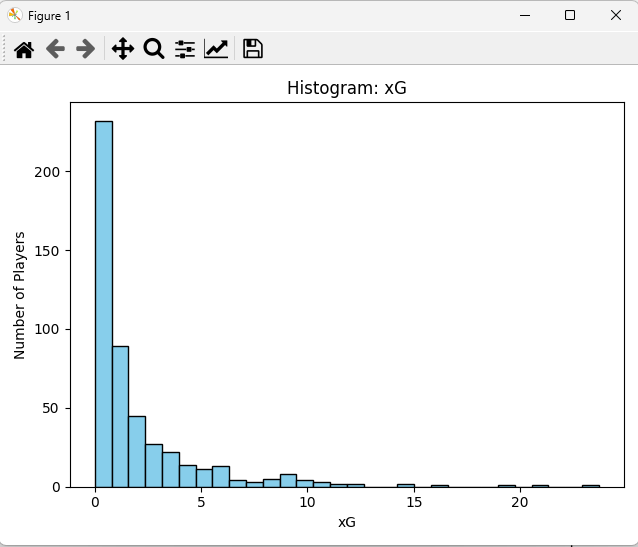
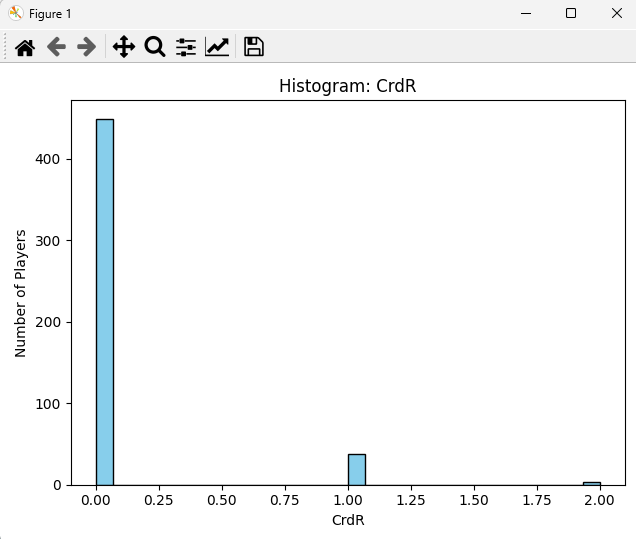
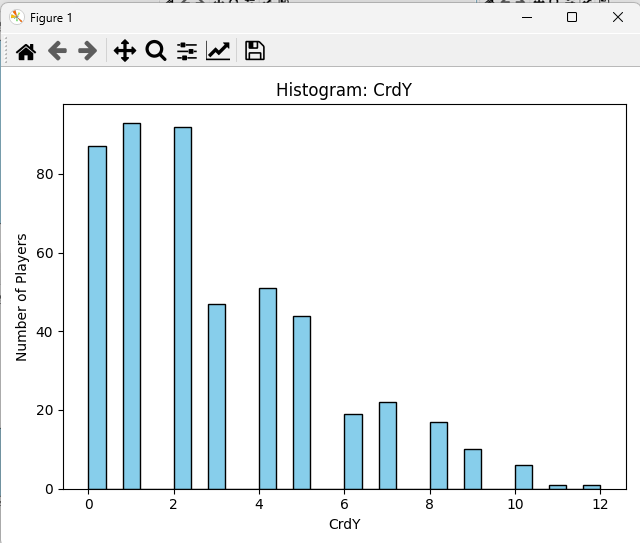
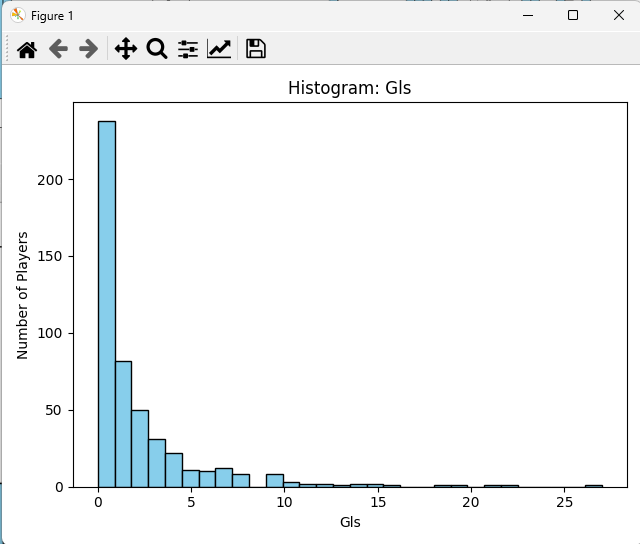
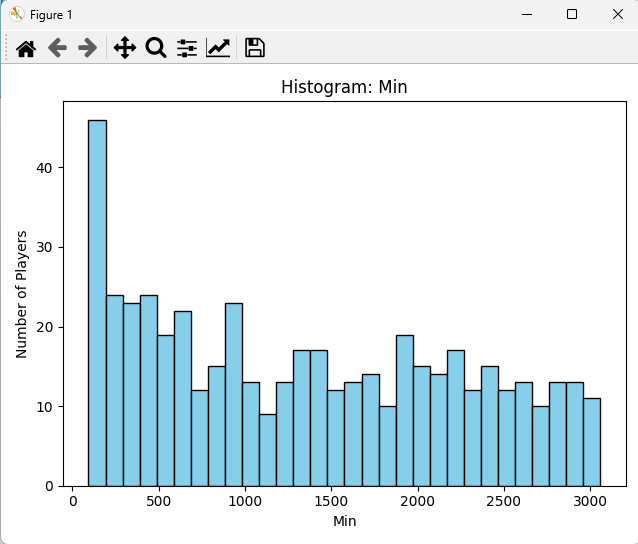
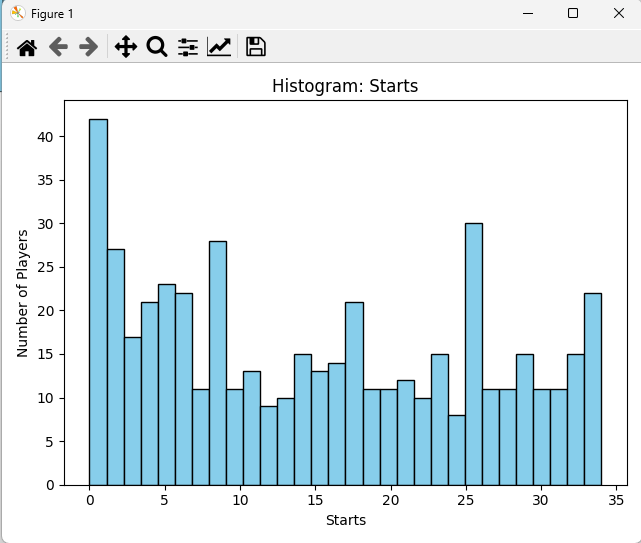
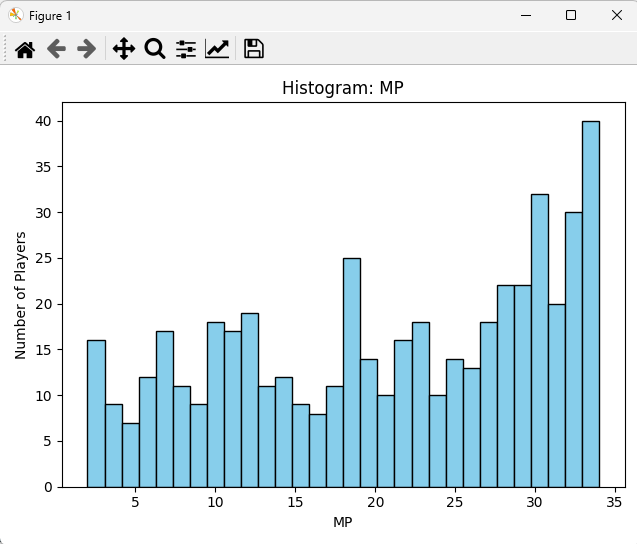
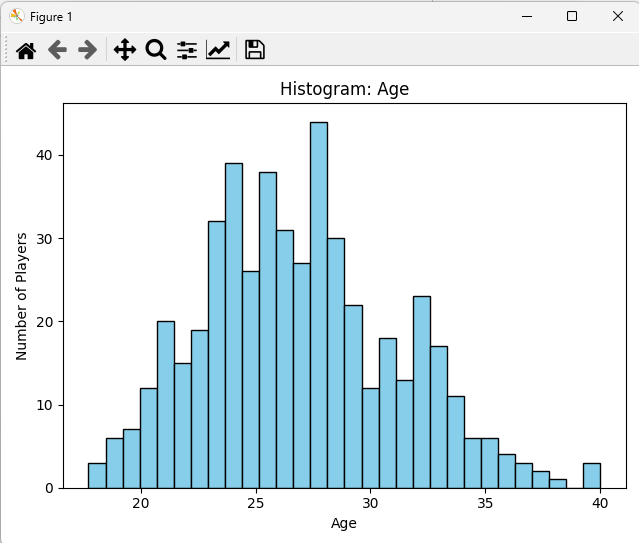
* **Identifying Numeric Columns: The code checks each column to identify which ones contain numeric data (ignoring non-statistic columns like 'Player', 'Nation', 'Squad', and 'Pos').** ****
* **Cleaning Data: It replaces 'N/a' with 0.0 and converts columns to the float type.** ****
* **Calculating Averages: It groups the data by 'Squad' and calculates the average for each numeric column.** ****
* **Saving Data: The calculated averages are saved to a new CSV file (ChiSoTeam.csv) **
* **Output:** ****
  + 1. **Technology Description**
* **pandas: For reading, cleaning, and manipulating tabular data efficiently**
* **matplotlib.pyplot: For creating histograms to visualize distributions of player statistics.**
* **csv, sys: Handle large CSV field sizes and system-level constraints.**
  + 1. **Code Breakdown**
* **The script reads player statistics from results.csv.** ****
* **Non-numeric placeholders like "N/a" are replaced with 0.00.** ****
* **The "Age" column, originally in the "years-days" format, is normalized into a decimal value (e.g., 20-200 → 20.55).** ****
* **All columns (except identifiers like "Player", "Nation", "Squad", "Pos") are converted to floats.**
* **Only numeric columns are used for histogram generation.** ****
* **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.** ****
  + 1. **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).**

****

* 1. **For all players in the league (Ve.py)**
     1. **Objective Overview**
* **The script processes football player data and generates histograms and bar charts to visualize player statistics and distributions across positions.**
  + 1. **Technology Description**
* **pandas: Used for loading, cleaning, and manipulating the dataset efficiently.**
* **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.**

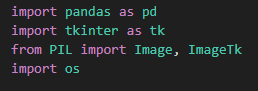
****

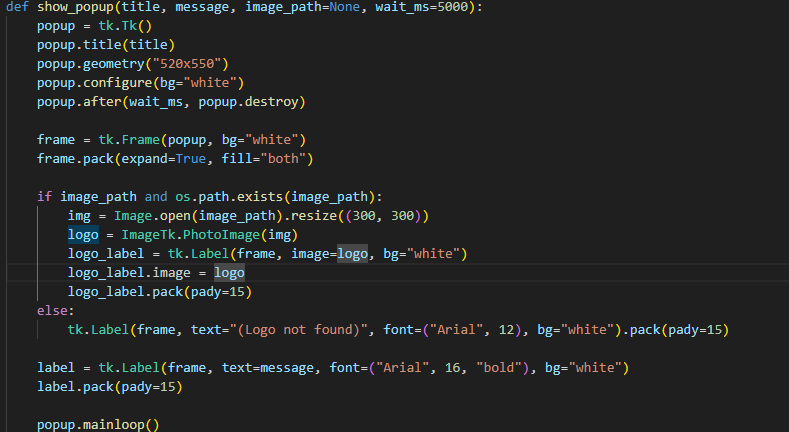
* + 1. **Code Breakdown**
* **The code starts by loading the results.csv dataset using pandas.** ****
* **Non-numeric values such as "N/a" are replaced with 0.00, and columns are converted to float types.** ****
* **The Age column is normalized from a "years-days" format to a decimal format (e.g., 20-200 → 20.55).** ****
* **For each numeric column (excluding identifiers), a histogram is plotted, representing the distribution of values.** ****
* **Additionally, a bar chart is created to display the number of players by position, based on the 'Pos' column.** ****
* **Any columns that cause errors during conversion or plotting are skipped.**
  + 1. **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).**

****

1. **Best Performing Team in the 2024-2025 Premier League season (DoiThanhTichTotNhat.py)**
   1. **Objective Overview**

* **Build a prediction system to determine the champion team based on season-long player statistics.**
  1. **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).**

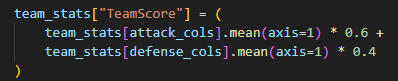
****

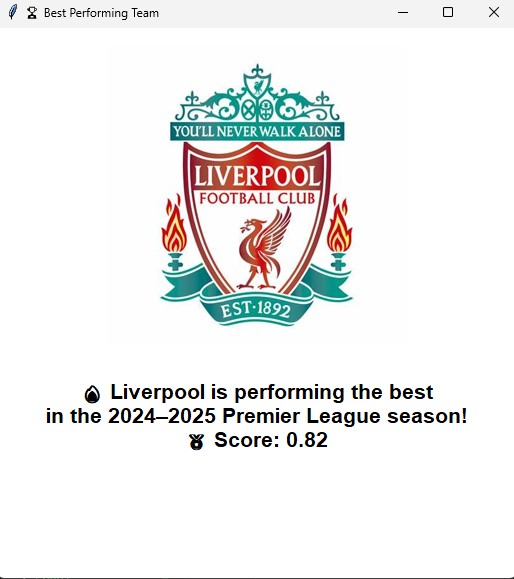
* 1. **Code Breakdown**
* **Shows initial playful popups with messages for certain teams to create anticipation.** ****
* **Reads data from results.csv, replaces "N/a" with 0.0, and fills any missing values with 0.0.**

****

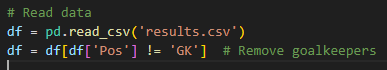
* **Two main stat categories: attack\_cols (e.g., Gls, Ast, xG) and defense\_cols (e.g., Tkl, Blocks, Save%).**

****

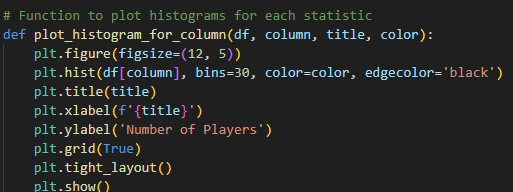
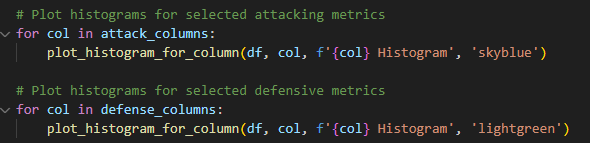
* **Averages player stats per team (groupby("Squad")).**
* **Normalizes all stats to a [0, 1] scale.**
* **Calculates a TeamScore using: TeamScore = 0.6 × Offensive Mean + 0.4 × Defensive Mean**
* **Identifies the team with the highest TeamScore**
  1. **Output**
* **A final GUI popup presents the predicted champion with the team logo and overall score.**

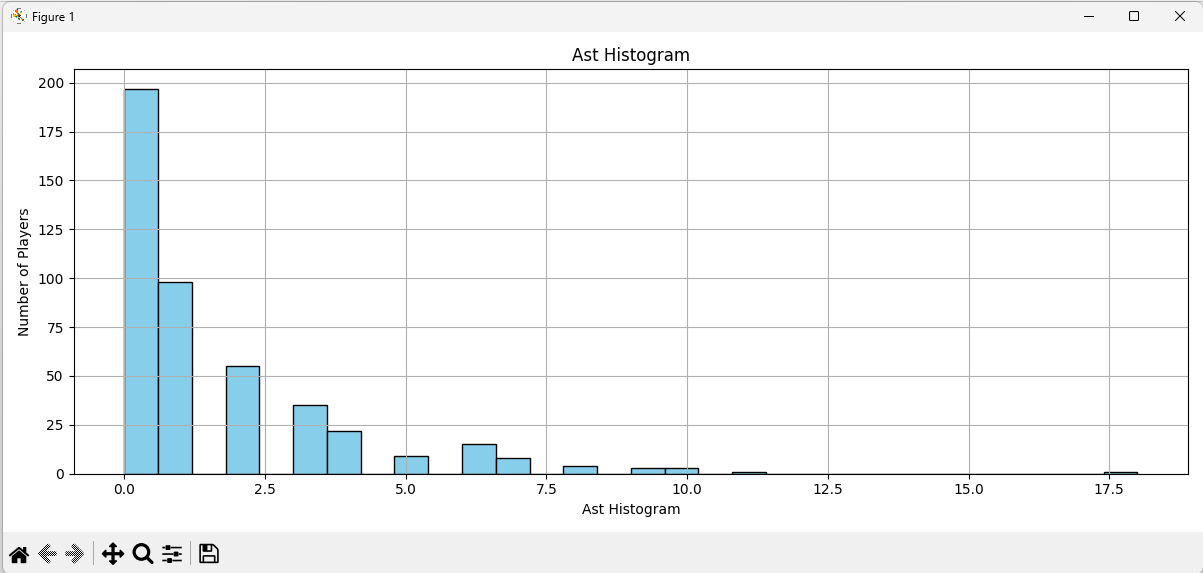
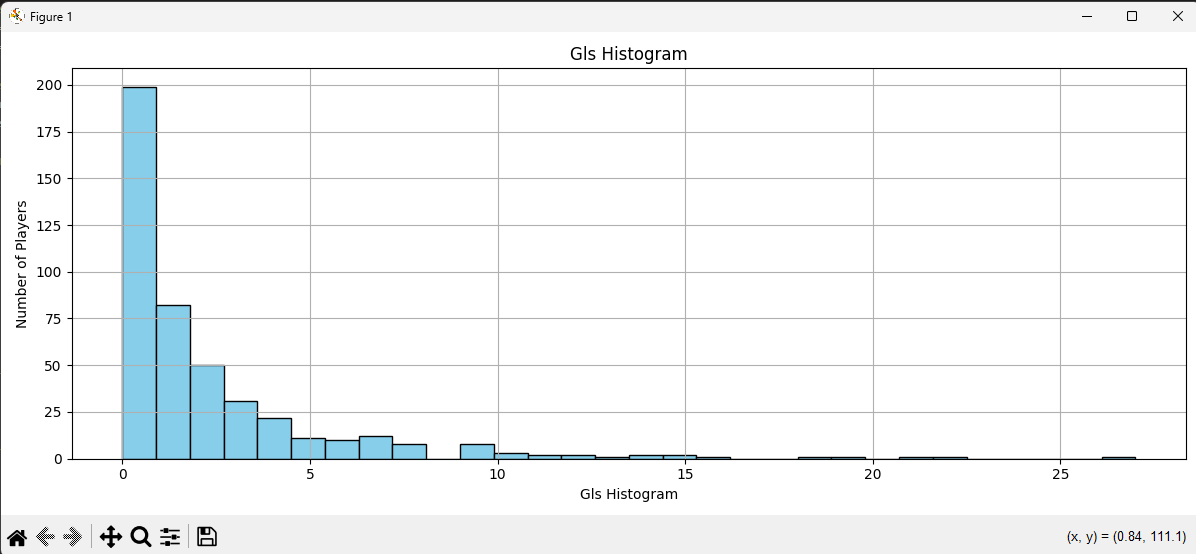
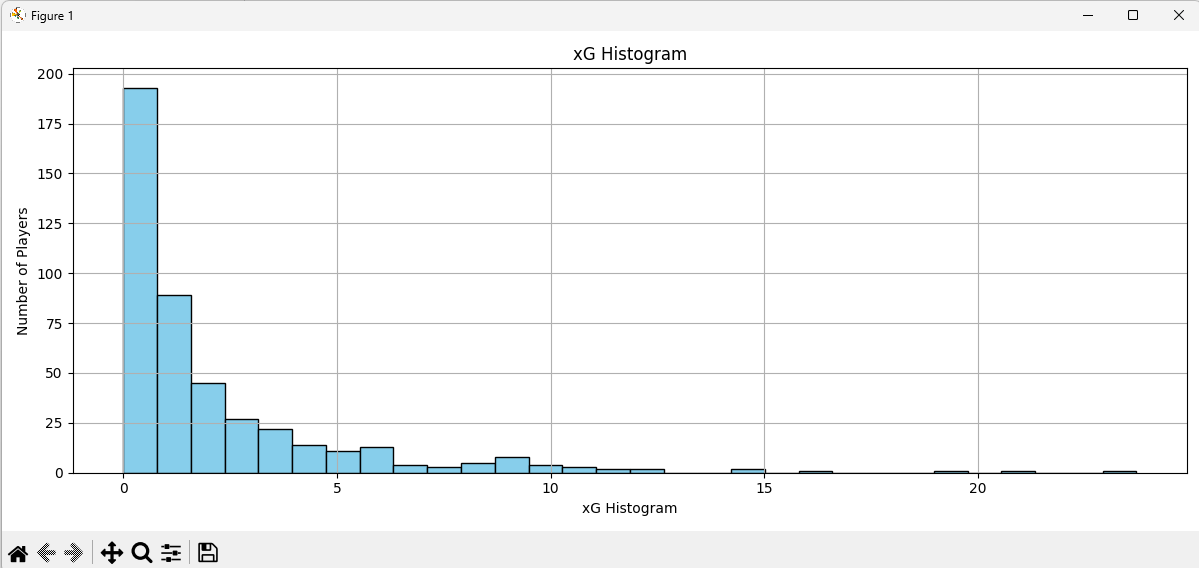
****

1. **Representative Offensive and Defensive Statistics(ATK\_DF.py)**
   1. **Objective Overview**

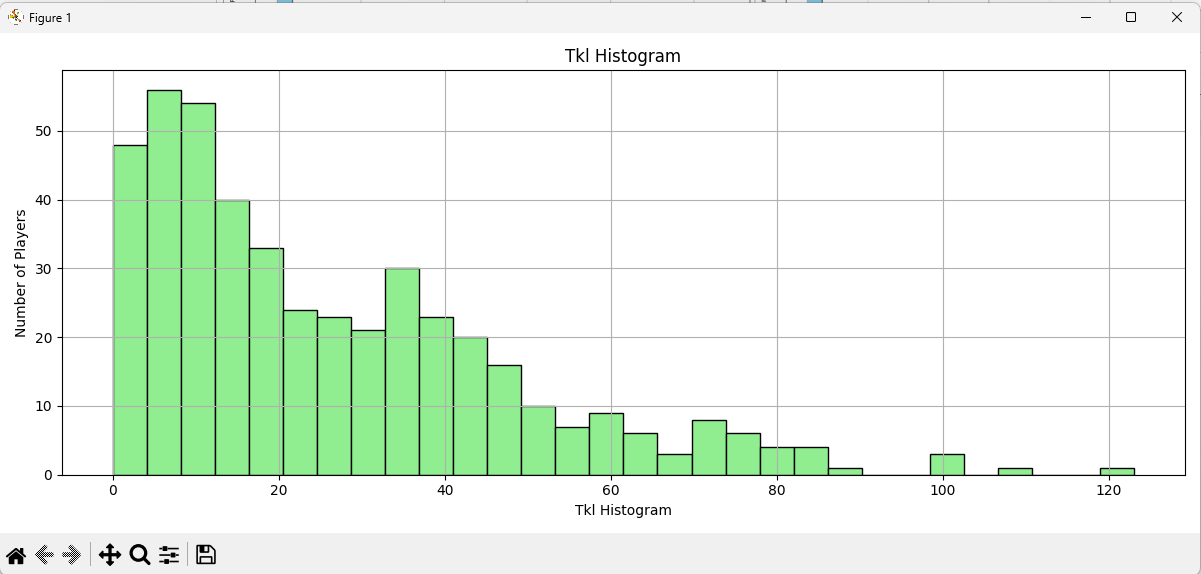
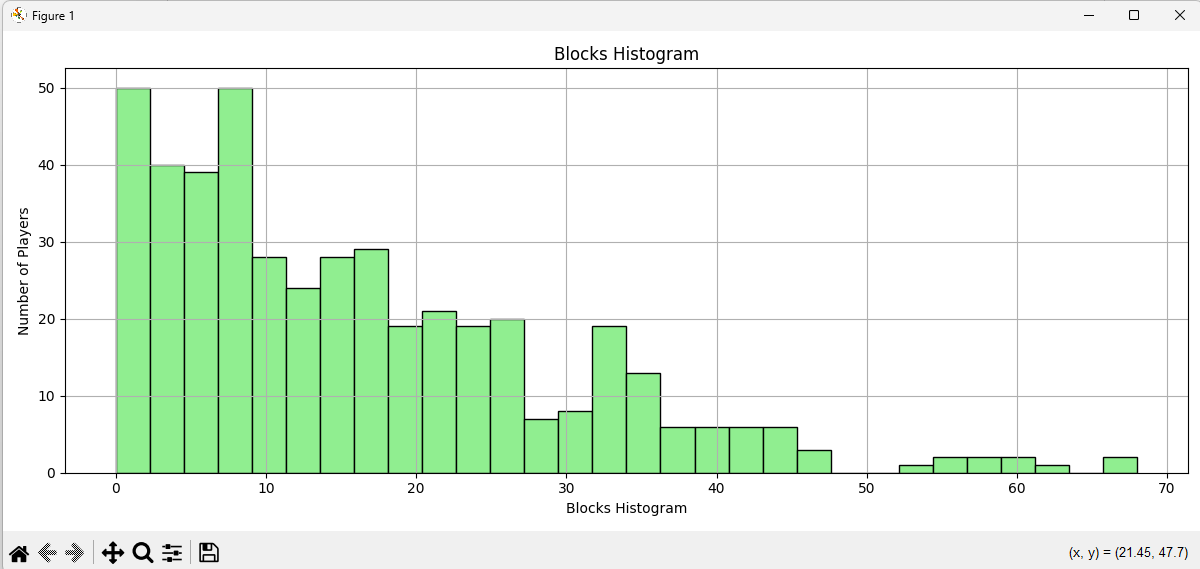
* **Objective: The goal is to visualize the distribution of specific football player statistics for attack and defense.**
* **Selected Metrics: We are focusing on three attacking metrics: Goals scored (Gls), Assists (Ast), and Expected Goals (xG); and three defensive metrics: Tackles (Tkl), Interceptions (Int), and Blocks (Blocks).**
  1. **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.** 
  1. **Code Breakdown**
* **The dataset results.csv is read into a DataFrame.**
* **Goalkeepers (Pos == 'GK') are removed from the dataset, focusing on outfield players.** ****
* **Attack Metrics: Goals (Gls), Assists (Ast), and Expected Goals (xG).**
* **Defense Metrics: Tackles (Tkl), Interceptions (Int), and Blocks (Blocks).**

****

* **A function plot\_histogram\_for\_column is created to generate a histogram for a given column, showing the distribution of player statistics.** ****
* **Histograms are plotted for each of the selected metrics (both attacking and defensive).** ****
  1. **Output**
* **Histograms for Attack Metrics:**
* **Goals (Gls), Assists (Ast), and Expected Goals (xG).**

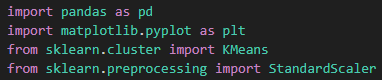
****

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

****

**Problem III**

1. **K-means algorithm (TimK.py)**
   1. **Find K (TimK.py)**
      1. **Objective Overview**

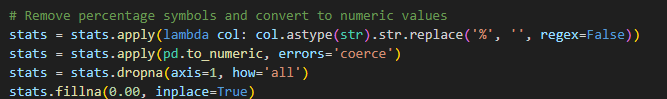
* **Use K-Means clustering to explore patterns in football player statistics.**
* **Determine the optimal number of clusters using the Elbow Method.**
  + 1. **Technology Description**
* **pandas: Loads and processes structured CSV data.**
* **matplotlib: Visualizes the Elbow Method to help select the best number of clusters.**
* **scikit-learn:**
* **StandardScaler: Standardizes numerical data.**
* **KMeans: Applies clustering to group players with similar statistical profiles.** 
  + 1. **Code Breakdown**
* **Loads results.csv.**

****

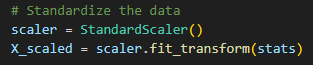
* **Selects all columns starting from 'Nation' onward for clustering**

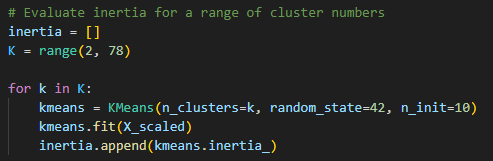
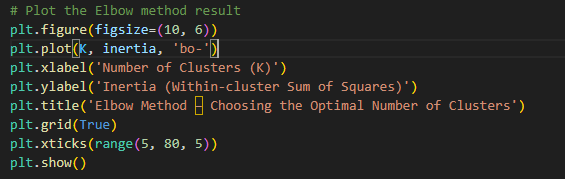
****

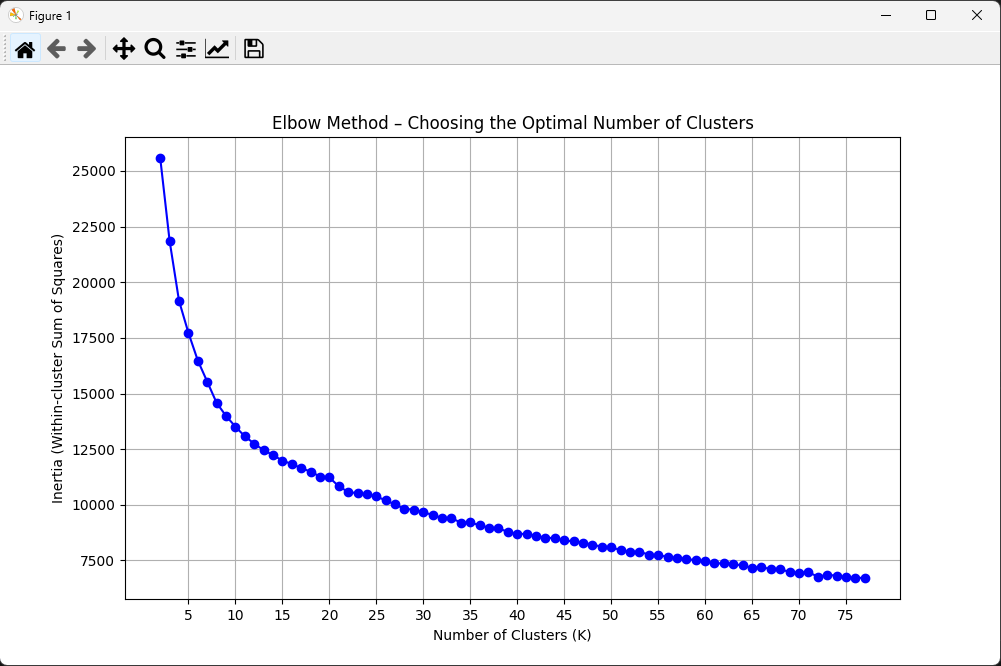
* **Removes percentage signs (%) from values.**
* **Converts strings to numeric values, coercing errors to NaN.**
* **Drops columns with all missing values, and fills the rest with 0.00.**

****

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

****

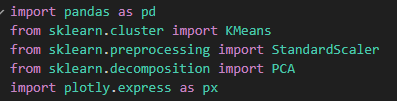
* **Tests multiple-cluster counts K from 2 to 77.** ****
* **For each K, calculates inertia (sum of squared distances within clusters).**
* **Plots inertia vs. number of clusters to identify the "elbow" point, which suggests the optimal number of clusters.** ****
  + 1. **Output**
* **A plot titled “Elbow Method – Choosing the Optimal Number of Clusters”.**
* **Helps visually identify the ideal K where inertia stops decreasing significantly.**

****

* + 1. **Answer the question**
* **How many groups should the players be classified 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) significantly slows down.**
* **Because diminishing returns after K = 10:**
* **Before K = 10, each additional cluster significantly improves how well the model fits 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).**

1. **Plot a 2D cluster of the data points (PhanLoaiCauThu.py)**
   1. **Objective Overview**

* **Use PCA to reduce the data dimensions to 2, then plot a 2D cluster of the data points.**
  1. **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.**

****

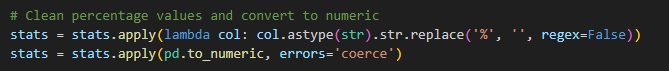
* 1. **Code Breakdown**
* **Read data from results.csv.**

****

* **Keep statistical columns starting from 'Nation' onward.**

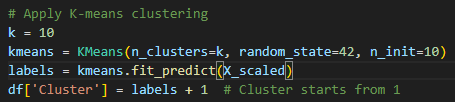
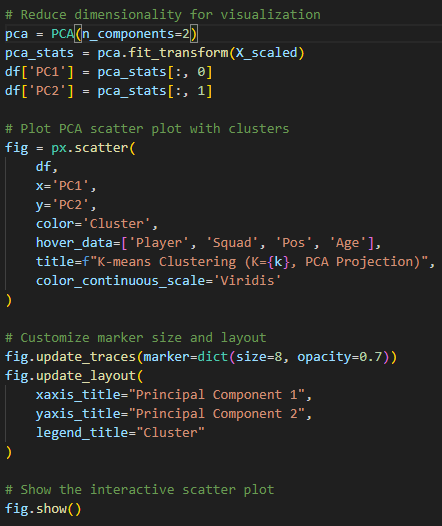
****

* **Remove % symbols and convert all data to numeric.**

****

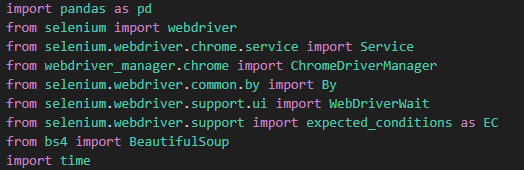
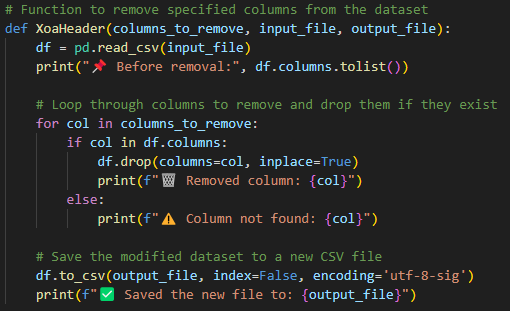
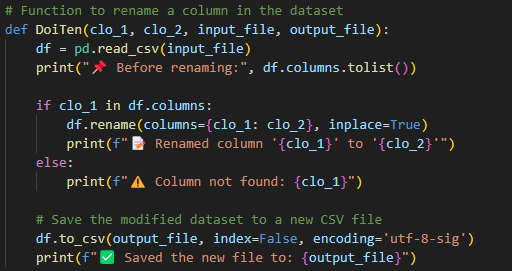
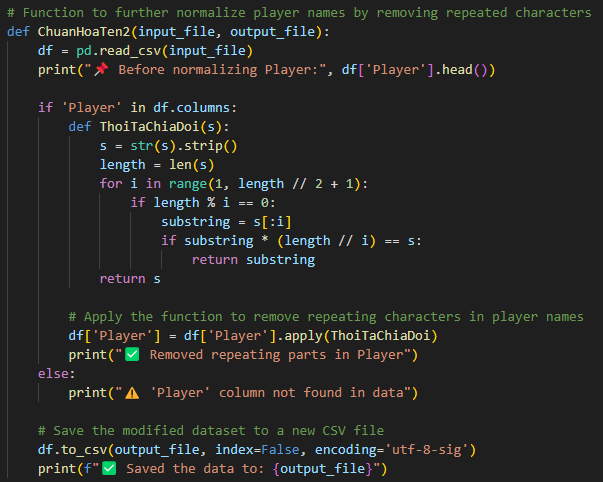
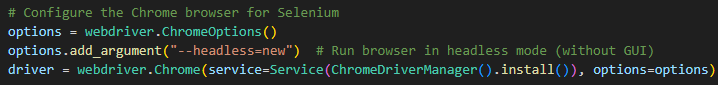
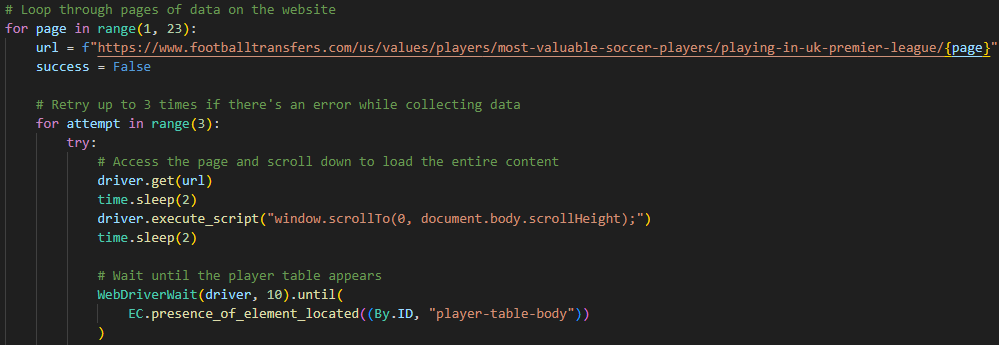
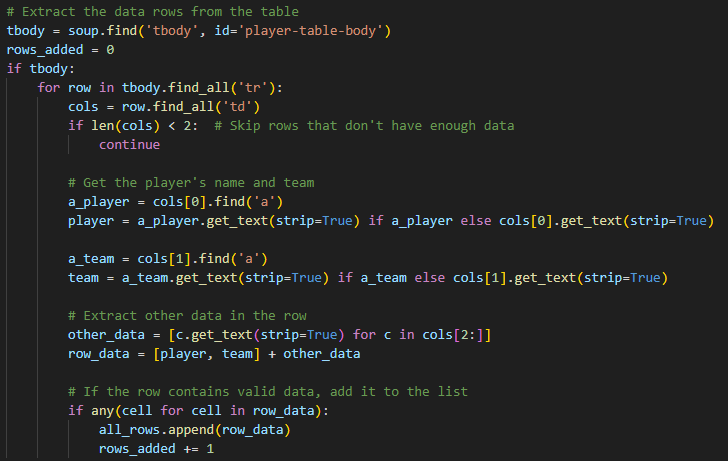
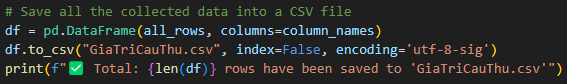
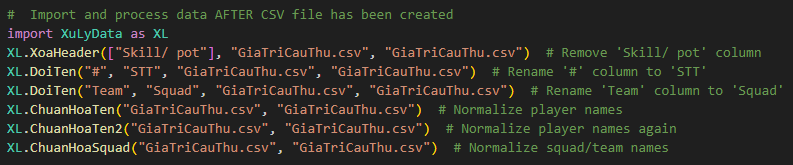
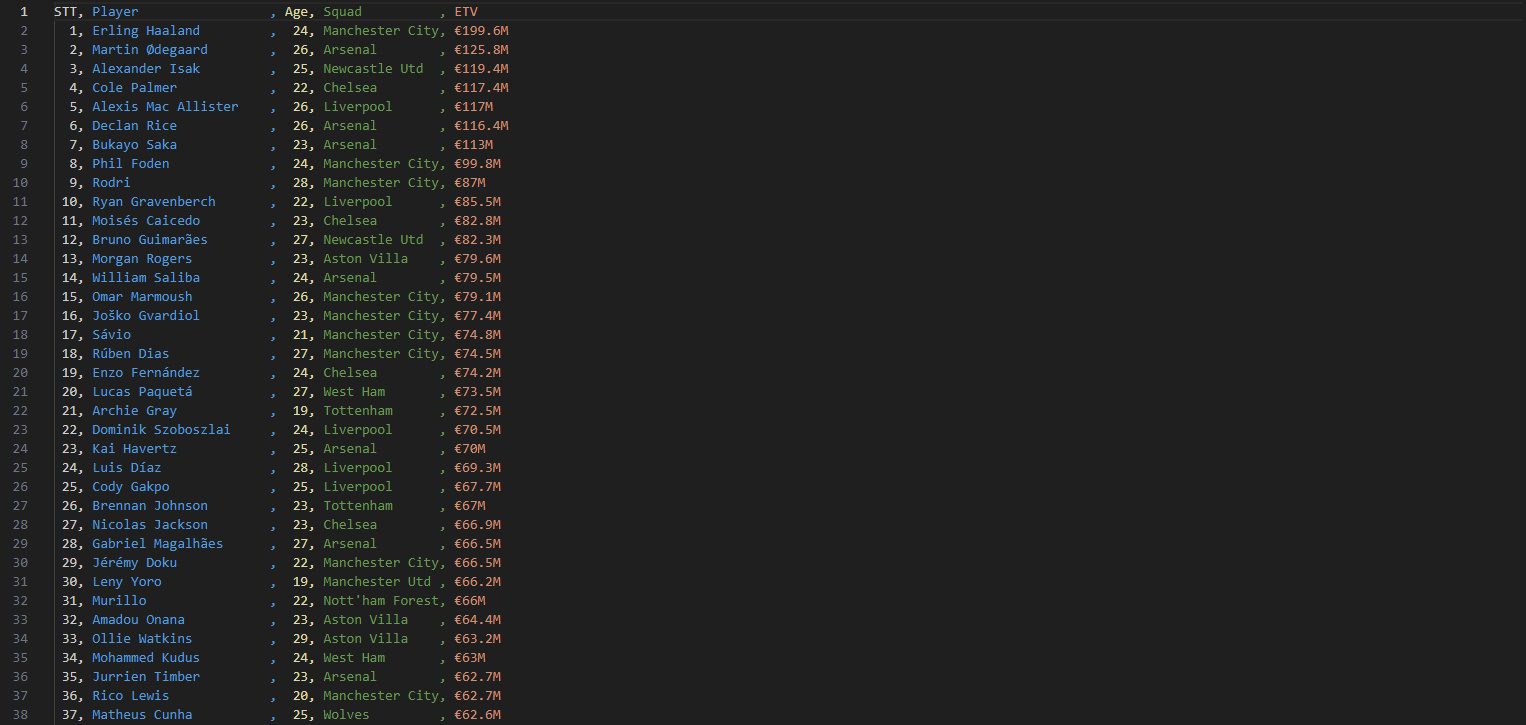
* **Fill any missing values with 0.**

****

* **Apply StandardScaler to normalize all features — a key step before clustering.** ****
* **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.** ****
  1. **Output**
* **An interactive 2D scatter plot where players are grouped into clusters (1–10) with different colors. Hovering over points shows player details, helping assess clustering effectiveness and group patterns.** ****

**Problem IV**

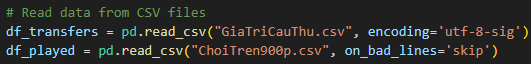
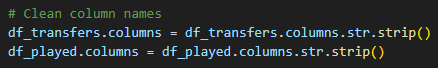
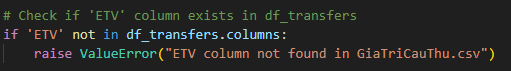
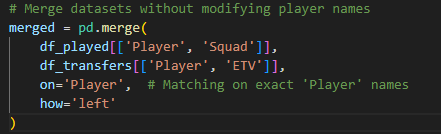
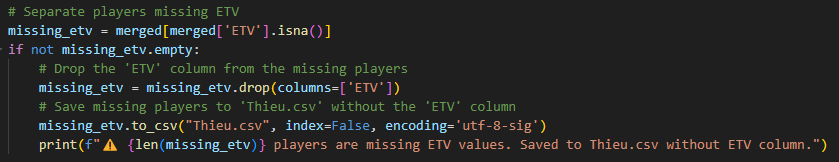
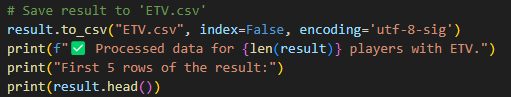
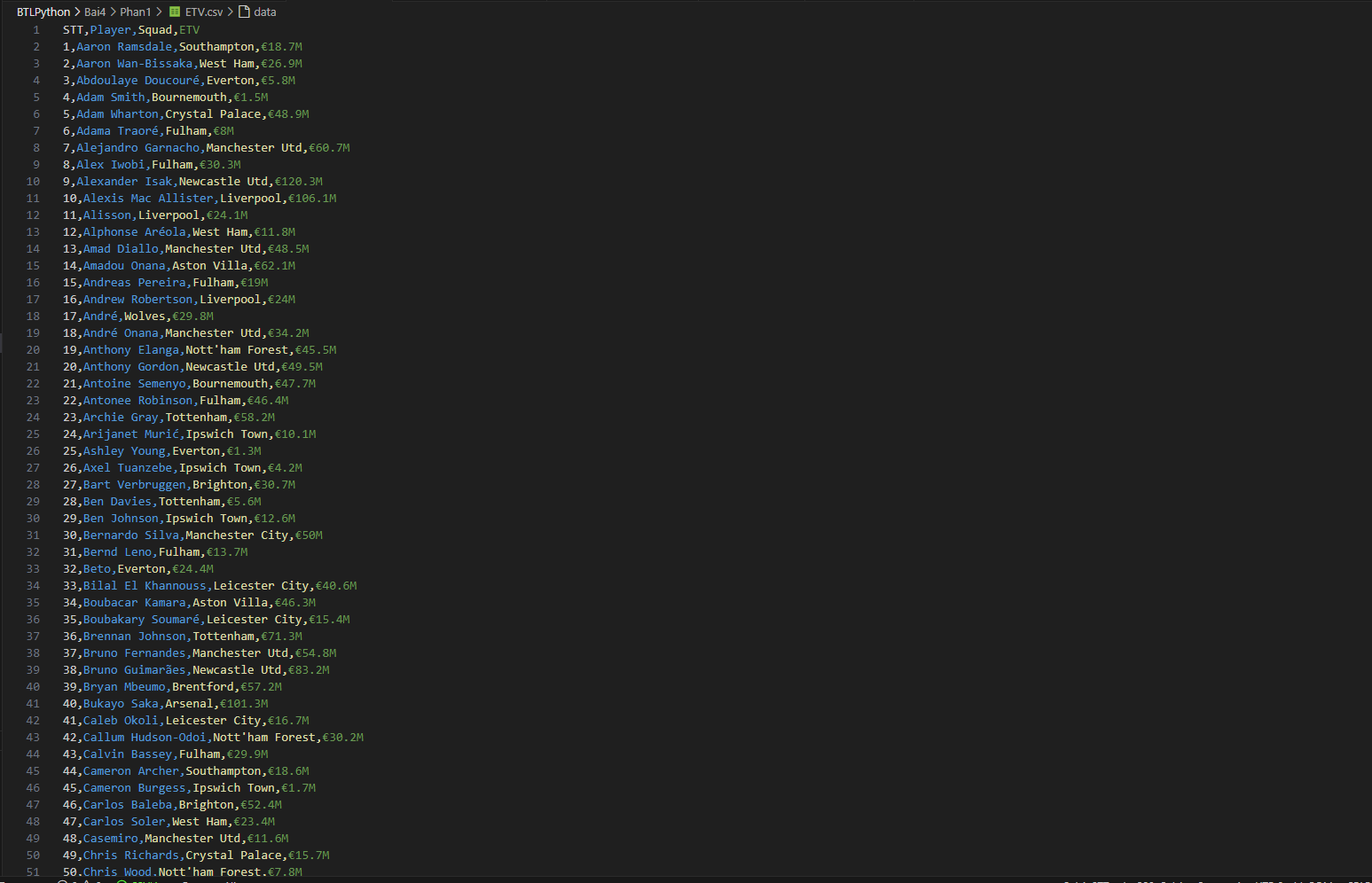
1. **Collect the transfer values of players for the 2024-2025 season from** [**https://www.footballtransfers.com/**](https://www.footballtransfers.com/) **whose playing time exceeds 900 minutes.**
   1. **Claw Data (LayData.py)**
      1. **Objective Overview**

* **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. **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 efficiency.**
* **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.** ****
* **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.** 
* **def DoiTen(): This function renames a column in the dataset. **
* **def ChuanHoaTen(): This function normalizes player names by removing any reference to the squad name from the player name column.** ****
* **def ChuanHoaTen2(): This function further normalizes player names by identifying and removing any repeated characters in the player names.** ****
* **def ChuanHoaSquad: This function normalizes squad names based on a predefined mapping. It updates squad names that may have variations or abbreviations.** ****
  + 1. **Code Breakdown**
* **The Chrome browser is configured in headless mode using Selenium.**
* **Webdriver Manager installs the correct ChromeDriver.** ****
* **The script loops through 22 pages, fetching data related to the players' names, teams, and stats from the player table (#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** ****
* **The collected data is saved into a Pandas DataFrame and then written to a CSV file (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.**
  + 1. **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.** ****
  1. **Merge Data (ETV.py & ETV2.py)**

**ETV**

* + 1. **Objective Overview**
    2. **Technology Description**
* **Pandas: Used for reading, cleaning, normalizing, and merging CSV data efficiently.**

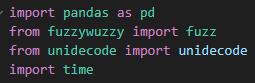
****

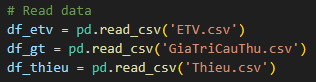
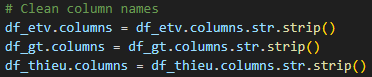
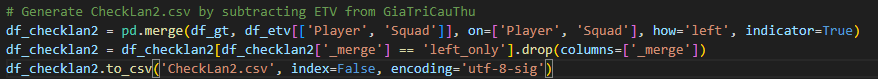
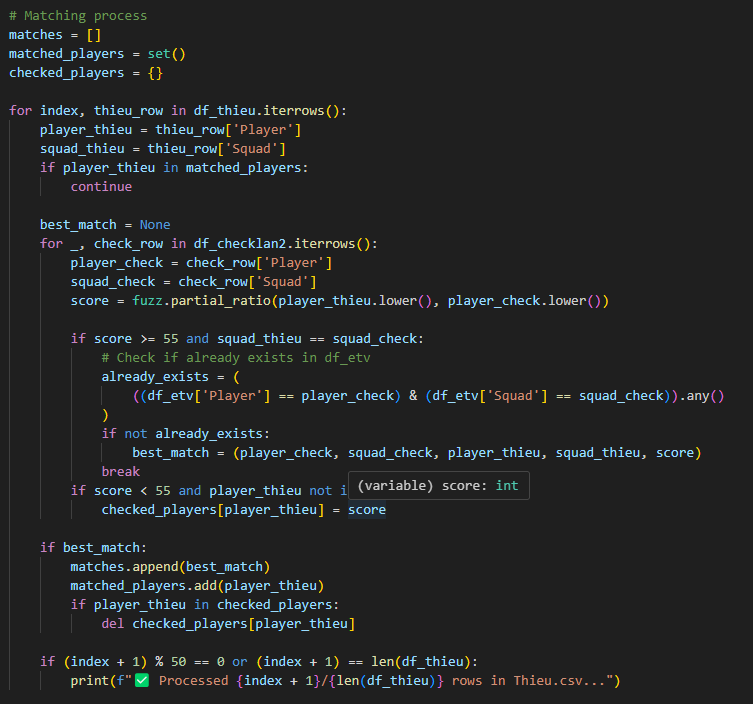
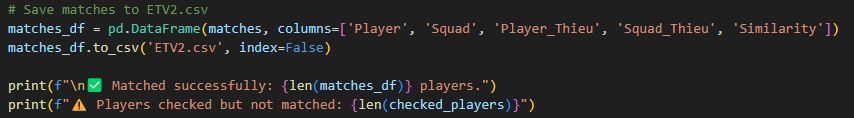
* + 1. **Code Breakdown**
* **Reads data from GiaTriCauThu.csv (ETV) and ChoiTren900p.csv (playing time >900 minutes).** 
* **Cleans column names by stripping leading and trailing spaces.** ****
* **Checks for the existence of the 'ETV' column in GiaTriCauThu.csv.** ****
* **Merges the datasets (df\_played and df\_transfers) on the 'Player' column, keeping only the matched players and their ETV values.** ****
* **Identifies and prints the list of players that did not match (missing ETV values).**
* **Saves unmatched players (missing ETV) to Thieu.csv.** ****
* **Filters out players without an ETV value and adds a sequential index.**
* **Saves the processed data with ETV to ETV.csv.** ****
  + 1. **Output**
* **ETV.csv: List of players with >900 minutes and their corresponding ETV values.** ****
* **Thieu.csv: List of players without ETV values (unmatched players).**

****

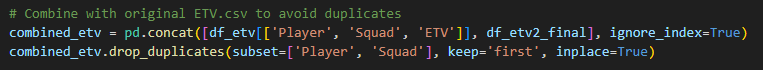
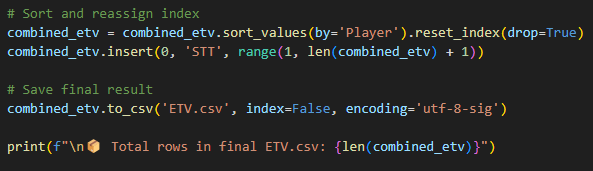
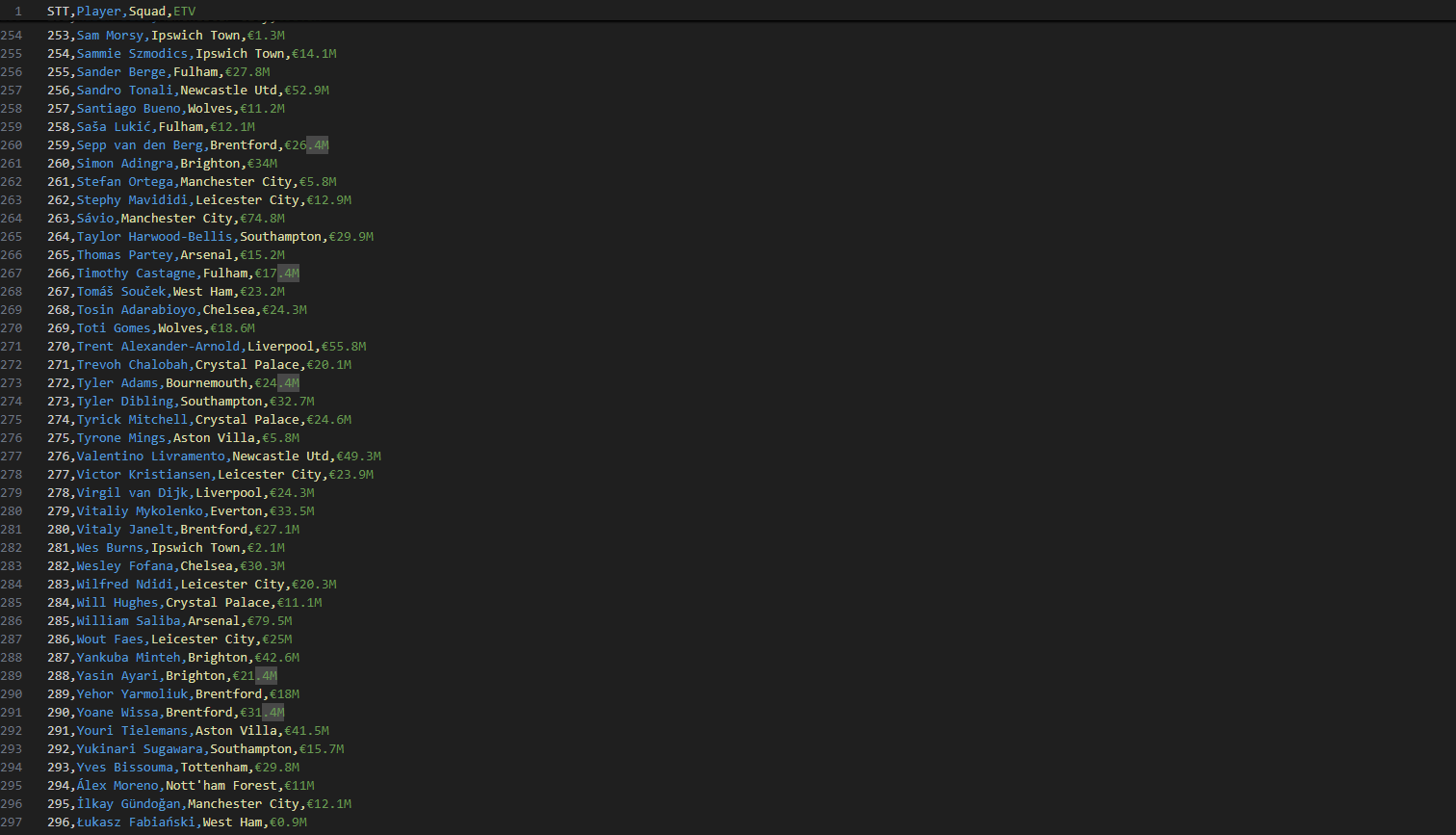
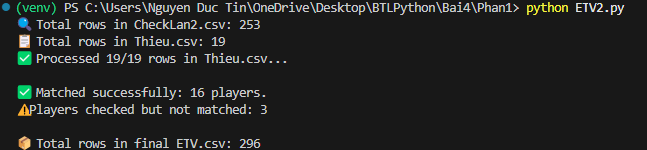
**ETV2**

* + 1. **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. **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.**

****

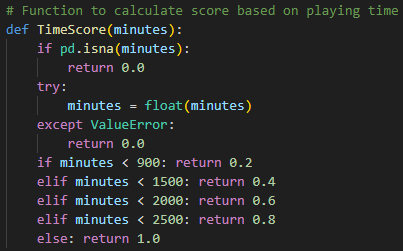
* + 1. **Code Breakdown**
* **Reads data from ETV.csv, GiaTriCauThu.csv, and Thieu.csv.** ****
* **Cleans column names to remove any extra spaces.** ****
* **Generates CheckLan2.csv by finding players from GiaTriCauThu.csv that are not in ETV.csv based on the "Player" and "Squad" columns.** ****
* **For 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**
* **All matched players with their similarity scores are saved to ETV2.csv.** ****
* **Merges the matches in ETV2.csv with GiaTriCauThu.csv to retrieve their ETV values.**

****

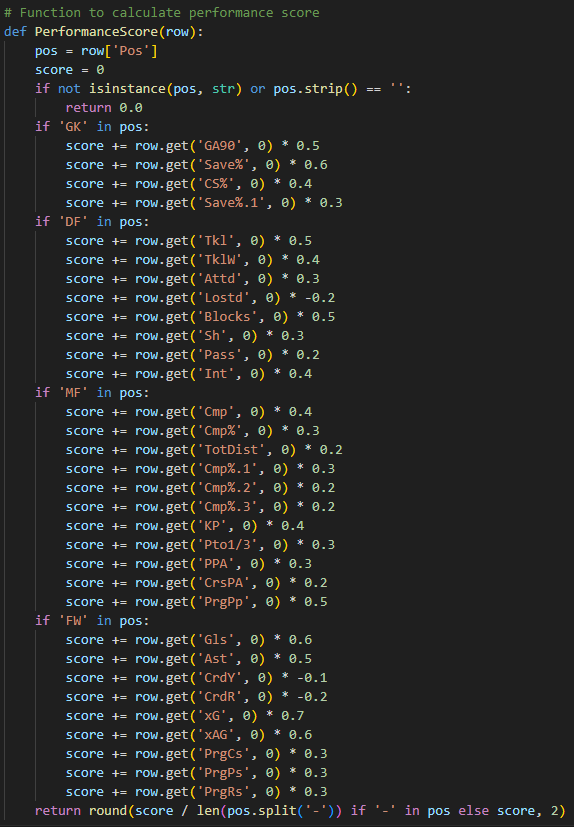
* **Combines the merged data with the original ETV.csv, ensuring no duplicates are present and sorting the data by player name.** ****
* **The final result is saved back to ETV.csv with a new index**
  + 1. **Output**
* **ETV2.csv: Contains matched players from Thieu.csv and CheckLan2.csv with their similarity scores.** ****
* **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).** ****
* **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**

1. **Propose a method for estimating player values.**
   1. **Make data for training**
      1. **Objective Overview**

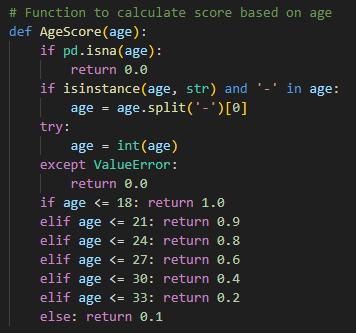
* **This script is designed to generate three key evaluation scores for football players—age score, playing time score, and performance score—based on match statistics and demographic information. The resulting data is saved to a file DataDanhGia.csv for downstream analysis or model training.**
  + 1. **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.** ****
  + 1. **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.**

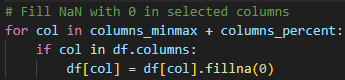
****

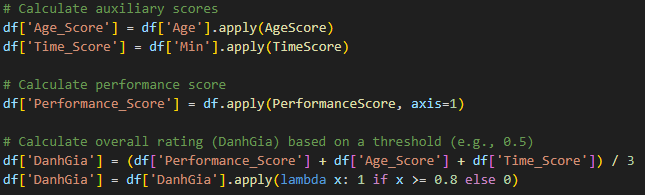
* **PerformanceScore(row): Calculates a weighted performance score based on the player’s position and their relevant metrics.  
  The weights vary depending on whether the player is a goalkeeper (GK), defender (DF), midfielder (MF), or forward (FW).  
  If a player has a hybrid position like DF-MF, the total score is divided accordingly.**

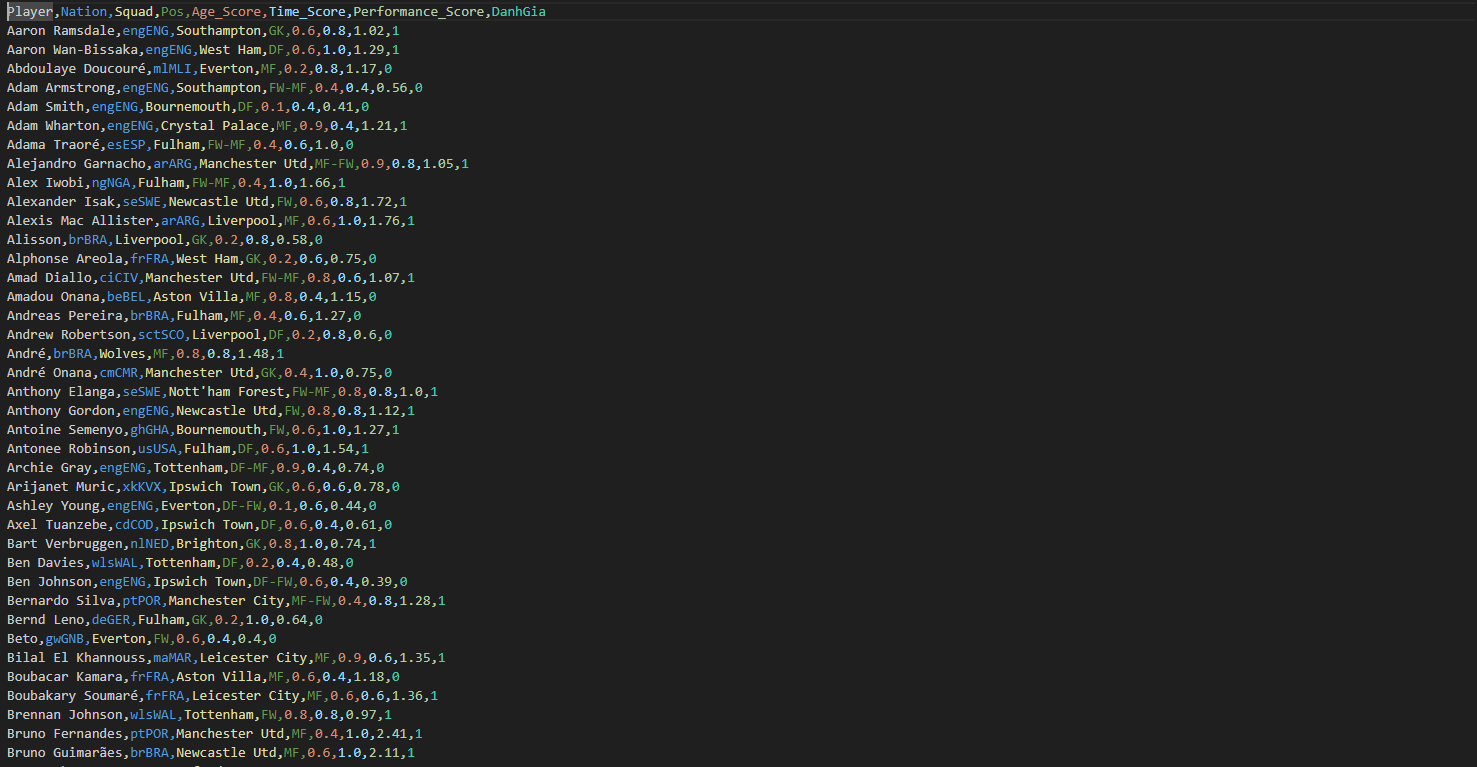
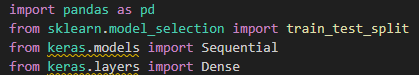
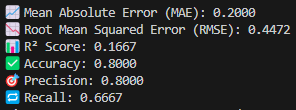
****

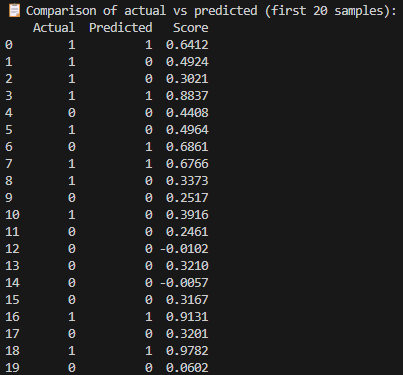
* **AgeScore(age): Converts a player's age into a normalized score, where younger players receive higher values. Special cases such as NaN or age in string format (e.g., "21-2") are handled gracefully.**
* **18 → 1.0**
* **19–21 → 0.9**
* **22–24 → 0.8**
* **25–27 → 0.6**
* **28–30 → 0.4**
* **31–33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

****

* **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.** ****
* **Percentage and performance columns are normalized using the two functions above.** ****
* **NaN values in relevant columns are replaced with 0 to ensure smooth computation during score calculation** ****
* **Calculated Age\_Score, Time\_Score, Performance\_Score, DanhGia**

****

* + 1. **Output**
* **This final dataset is saved as DataDanhGia.csv for further use in analysis or modeling.** 
  1. **Training model (TrainModel.py)**
     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.**
  + 1. **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 efficient gradient-based training.**
* **Metrics: Uses Mean Absolute Error (MAE) to monitor model performance during training. **
  + 1. **Code Breakdown**
* **The CSV file DataDanhGia.csv is read using pandas.** ****
* **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.** ****
* **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.**
  + 1. **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** 

****

* + 1. **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.**

1. **Model Selection:**

* **Start with Linear Regression (simple, interpretable).**
* **Use more complex models like Random Forest or Gradient Boosting if needed for non-linear patterns.**

1. **Preprocessing:**

* **Normalize features and handle missing values to improve model accuracy.**

1. **Training and Evaluation:**

* **Split data into Train, Validation, and Test sets.**
* **Evaluate using MAE, RMSE, and R².**

1. **Deployment**

* **After training, use the model to predict player values and analyze feature importance.**