Posts and Telecommunications
Institute of Technology

Report on Project Assignment 1

PYTHON PROGRAMMING LANGUAGE

Supervisor:	Kim Ngoc Bach
Student:	Chien Dang
Student ID:	B23DCCE012
Class:	D23CQCE06-B
Academic Term:	2023 – 2028

TABLE OF CONTENTS

PROBLEM 1

1.Objectives

Main Objective: Develop a Python program to collect detailed statistical data for players with >90 minutes played in the 2024-2025 English Premier League from FBref.com and save it to a CSV file named 'results.csv'.

Specific Objectives:

- Collect basic player information (Nation, Team, Position, Age).
- Gather Playing Time statistics (Matches Played, Starts, Minutes).
- Gather Performance metrics (Goals, Assists, Yellow Cards, Red Cards).
- Gather Expected metrics (xG, xAG).
- Gather Progression metrics (PrgC, PrgP, PrgR).
- Gather Per 90 minutes metrics (Gls, Ast, xG, xGA).
- Gather Goalkeeping data (GA90, Save%, CS%, Penalty Saves).
- Gather Shooting data (SoT%, SoT/90, G/Sh, Avg. Shot Distance).
- Gather Passing data (Total, Short, Medium, Long: Passes, Completion %, Progressive Distance; Key Passes, Final Third Passes, Penalty Area Passes, Crosses into Penalty Area, PrgP).
- Gather Goal and Shot Creation data (SCA, GCA: Count, Per 90 minutes).
- Gather Defensive Actions data (Tackles, Challenges, Blocks).
- Gather Possession data (Touches, Take-Ons, Carries, Receiving).
- Gather Miscellaneous Stats (Fouls, Fouled, Offsides, Crosses, Recoveries, Aerial Duels).
- 'results.csv' file structure: Columns as stats, players sorted alphabetically by name, missing data marked as "N/a".

2. Methodology

Tool and Library Selection:

Selenium: Employed for automating the Chrome web browser. Selenium empowers the program to interact with the website as a human user, overcoming challenges such as JavaScript execution and dynamic data loading. This is crucial because FBref uses JavaScript to render its data.

WebDriverManager: Facilitates the automatic management of the Chrome browser driver, ensuring the driver version is compatible with the installed Chrome version.

Beautiful Soup: Utilized to parse the HTML structure of the webpage after it has been loaded by Selenium. Beautiful Soup simplifies the process of searching for and extracting data from HTML tags.

Pandas: Utilized to create and manage DataFrames, which are powerful table-like data structures. Pandas facilitates the efficient storage, processing, and exporting of data to CSV files.

Program Design:

Initialization:

Import necessary libraries.

Define constants: TABLE_LINKS (table URLs and IDs), PLAYER_KEYS (player data keys).

Functional Modules (Functions):

initialize_player_dict(): Creates a dictionary with player data keys, defaulting values to "N/a."

scrape_standard_stats(): Scrapes core player data from the FBref "Standard Stats" table.

update_goalkeeping_stats(player_set): Adds goalkeeping statistics to player data.

update_shooting_stats(player_set): Adds shooting statistics to player data.

update_passing_stats(player_set): Adds passing statistics to player data.

update_goal_shot_creation_stats(player_set): Adds goal and shot creation statistics to player data.

update_defensive_stats(player_set): Adds defensive actions statistics to player data.

update_possession_stats(player_set): Adds possession statistics to player data.

update_miscellaneous_stats(player_set): Adds miscellaneous statistics to player data.

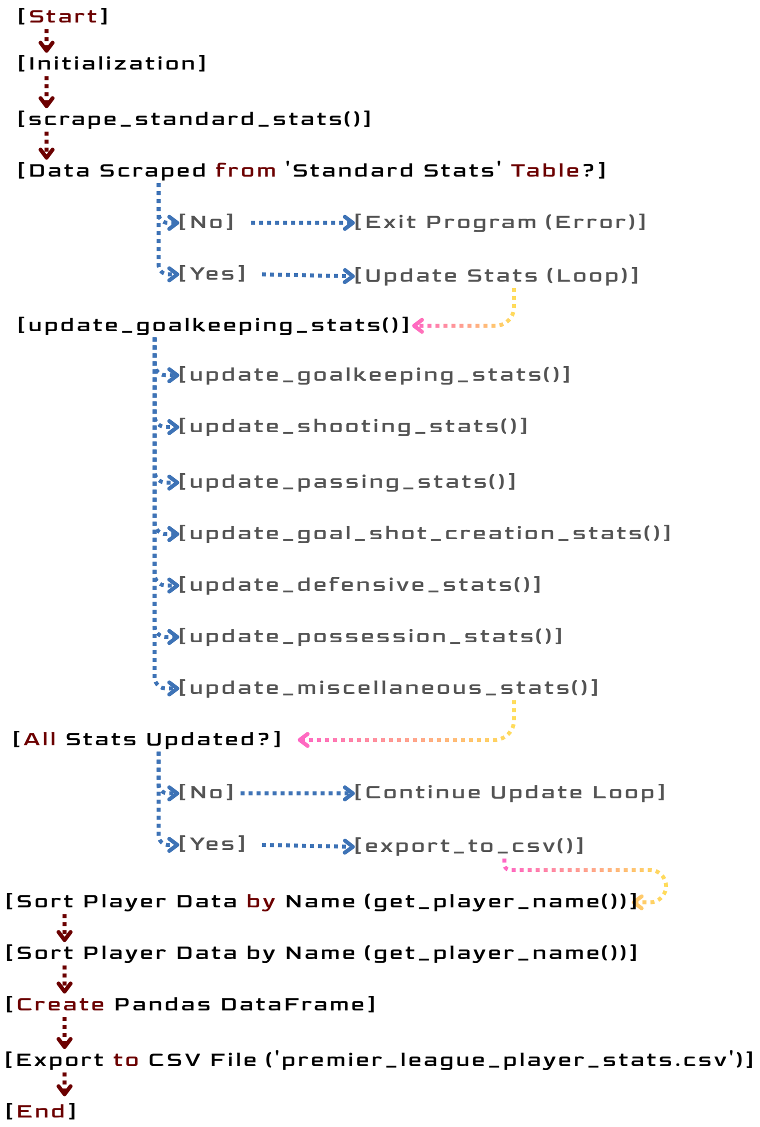
get_player_name(player_dict): Retrieves a player's name from their data for sorting.

format_player_data(player_dict): Formats player data into a consistent list for CSV export.

export_to_csv(player_set_dict): Exports player data to a CSV file, sorted by name.

main() Function: Orchestrates data scraping, updating, and exporting processes.

3.Code Execution Flowchart


[Start]
[Initialization]
[scrape_standard_stats()]
[Data Scraped from 'Standard Stats' Table?]
[No]
[Exit Program (Error)]
[Yes]
[Update Stats (Loop)]
[update_goalkeeping_stats()]
[update_goalkeeping_stats()]
[update_shooting_stats()]
[update_passing_stats()]
[update_goal_shot_creation_stats()]
[update_defensive_stats()]
[update_possession_stats()]
[update_miscellaneous_stats()]
[All Stats Updated?]
[No]
[Continue Update Loop]
[Yes]
[export_to_csv()]
[Sort Player Data by Name (get_player_name())]
[Sort Player Data by Name (get_player_name())]
[Create Pandas DataFrame]
[Export to CSV File ('premier_league_player_stats.csv')]
[End]

4.Source Code

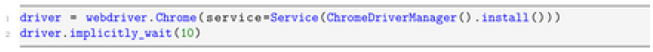
https://github.com/duahauhz/ASSIGNMENT_01_PYTHON_PTIT/blob/master/RESULTS/TASK%201/assignment_01_task_01.py

PROBLEM 1

5. Explain The Code

Approach:
Specialized Separation:
Each type of statistic (defense, offense, etc.) is handled by a dedicated function, ensuring modularity and ease of maintenance.
Selenium + BeautifulSoup Combination:
Selenium simulates browser behavior to load dynamic pages.
BeautifulSoup parses the static HTML, optimizing for speed and stability.

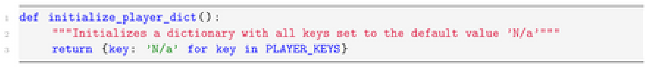
WebDriver Management:



```
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))
driver.implicitly_wait(10)
```

Rationale:
FBref uses JavaScript to render its tables. Therefore, a virtual browser is required to fully load the content.
WebDriverManager automatically downloads a compatible ChromeDriver version, preventing version mismatch errors.
Specifications:
implicitly_wait(10): Sets a default timeout of 10 seconds. This balances waiting for sufficient data loading with preventing system hangs.

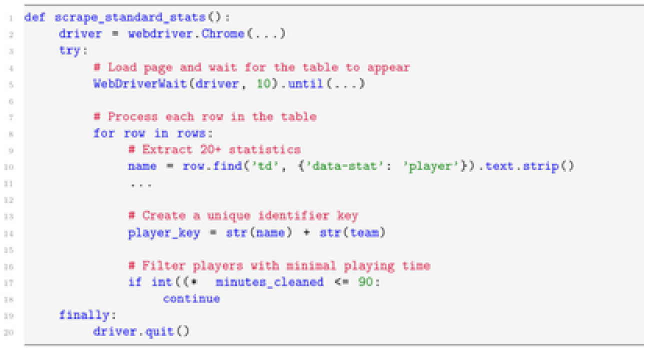
Player Dictionary Initialization Function:



```
def initialize_player_dict():
    """Initializes a dictionary with all keys set to the default value 'N/a'"""
    return {key: 'N/a' for key in PLAYER_KEYS}
```

Rationale:
Ensures that every player has a consistent data structure from the outset.
Handles missing data by providing default values.
Prevents KeyError exceptions when accessing attributes that have not yet been updated.

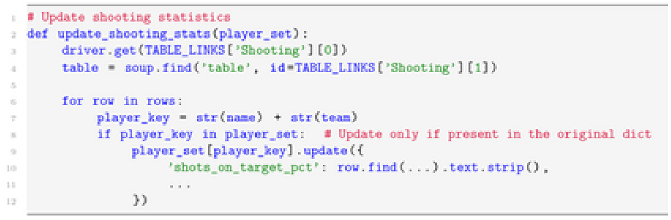
Data Collection Function (Standard Stats):



```
def scrape_standard_stats():
    driver = webdriver.Chrome(...)
    try:
        # Load page and wait for the table to appear
        WebDriverWait(driver, 10).until(...)

        # Process each row in the table
        for row in rows:
            # Extract 20+ statistics
            name = row.find('td', {'data-stat': 'player'}).text.strip()
            ...

            # Create a unique identifier key
            player_key = str(name) + str(team)

            # Filter players with minimal playing time
            if int((*  minutes_cleaned <= 90:
                continue
    finally:
        driver.quit()
```

Step-by-Step Explanation:
Initialize a virtual browser to load the dynamic webpage.
Wait for the table to load with a 10-second timeout (to prevent errors due to slow network connections).
Process each data row:
    Use find() with data-stat to retrieve each specific statistic accurately.
    Normalize the data immediately upon extraction (using strip()).
Filtering mechanism:
    Skip players with ≤ 90 minutes of playing time → Remove noisy data.
Close the browser within the finally block → Ensure resource release.

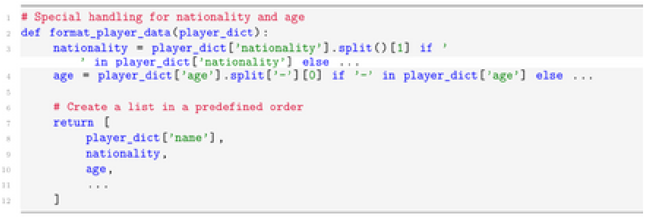
Advanced Statistics Update Functions:
(Example with update_shooting_stats())



```
# Update shooting statistics
def update_shooting_stats(player_set):
    driver.get(TABLE_LINKS['Shooting'][0])
    table = soup.find('table', id=TABLE_LINKS['Shooting'][1])

    for row in rows:
        player_key = str(name) + str(team)
        if player_key in player_set:  # Update only if present in the original dict
            player_set[player_key].update({
                'shots_on_target_pct': row.find(...).text.strip(),
                ...
            })
```

Design Characteristics:
Reuses player_key: Ensures consistency with the original data.
Checks for existence before updating: Prevents the creation of unintended new records.
Updates individual statistics instead of overwriting the entire record: Preserves existing data.
Reasons for Separation into Multiple Functions:
Each type of statistic has a different URL and HTML structure.
Facilitates easy maintenance/addition of individual statistic types.
Optimizes performance (potentially allowing for parallel execution in the future).

Output Data Formatting Function:



```
# Special handling for nationality and age
def format_player_data(player_dict):
    nationality = player_dict['nationality'].split()[1] if '
        ' in player_dict['nationality'] else ...
    age = player_dict['age'].split['-'][0] if '-' in player_dict['age'] else ...

    # Create a list in a predefined order
    return [
        player_dict['name'],
        nationality,
        age,
        ...
    ]
```

Processing Workflow:
Special field preprocessing:
    Nationality: "fr France" → "France"
    Age: "25-1998" → "25"
Mapping according to the column order defined in export_order_keys.
Ensures consistent column order in the CSV output.

6. Results

The output is a results.csv file.

PROBLEM 2

1.Objectives

Top Player Analysis:
Identify the 3 players with the highest values for each statistic.
Identify the 3 players with the lowest values for each statistic.
Save the results to the 'top_3.txt' file in a clear and easily readable format.
Statistical Analysis:
Calculate basic statistical measures for each statistic:
Median for the entire league and for each team.
Mean for the entire league and for each team.
Standard deviation for the entire league and for each team.
Save the results to the 'results2.csv' file with a well-defined structure.
Team-Based Analysis:
Determine the team with the highest average value for each statistic.
Assess the overall strongest team based on a composite of the different statistics.
Analyze the relative performance between the teams.
Data Visualization:
Generate histograms for each statistic to visualize its distribution.
Create violin plots combined with swarm plots to display the distribution of the data.
Highlight the top 3 teams with the best performance.
Customize the graphical interface for a professional look and feel.

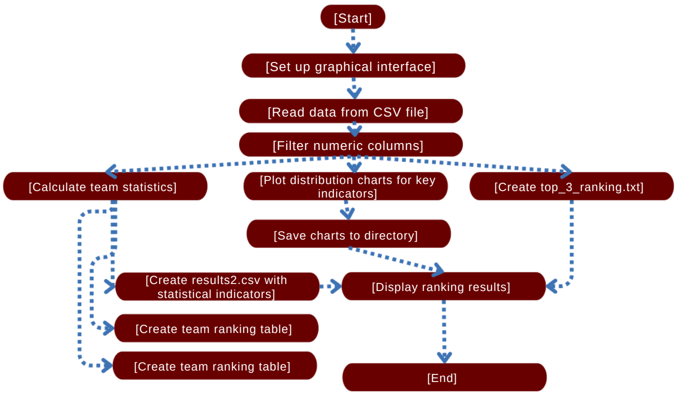
2. Methodology

Data Handling:
Employ the Pandas library to read, clean, and process the data from CSV files.
Statistical Analysis:
Calculate statistical measures, including median, mean, and standard deviation, for each statistic both overall and by team.
Visualization:
Utilize Matplotlib and Seaborn to generate violin plots combined with swarm plots for data distribution visualization.
Ranking:
Evaluate team performance based on a composite of key performance indicators.
Results Storage:
Export the analysis reports as both text and CSV files.

3.Source Code

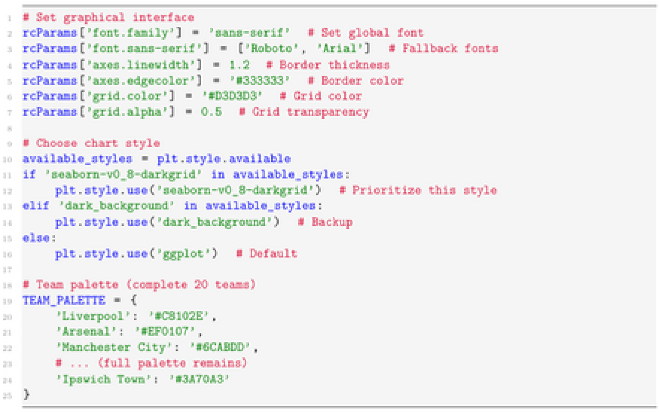
https://github.com/duahauhz/ASSIGNMENT_01_PYTHON_PTIT/blob/master/RESULTS/TASK%202/assignment_01_task_02.py

4.Code Execution Flowchart


[Start]
[Set up graphical interface]
[Read data from CSV file]
[Filter numeric columns]
[Calculate team statistics]
[Plot distribution charts for key indicators]
[Create top_3_ranking.txt]
[Save charts to directory]
[Create results2.csv with statistical indicators]
[Display ranking results]
[Create team ranking table]
[Create team ranking table]
[End]

5. Explain The Code

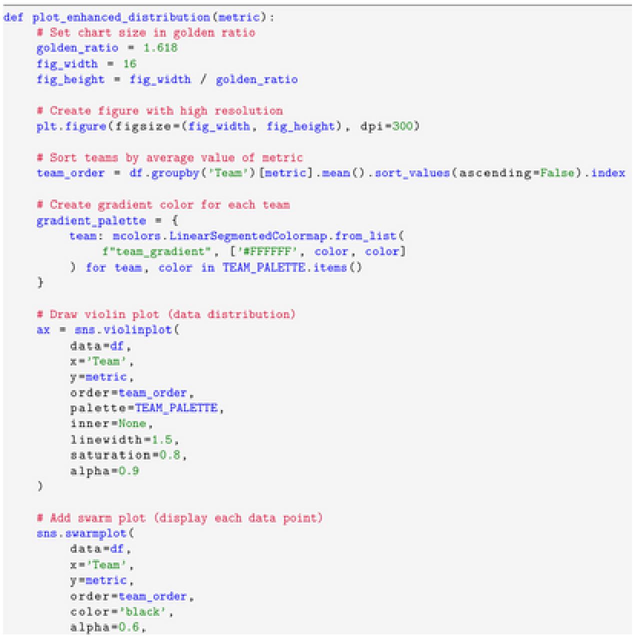
Initial Setup:



```python
# Set graphical interface
rcParams['font.family'] = 'sans-serif'  # Set global font
rcParams['font.sans-serif'] = ['Roboto', 'Arial']  # Fallback fonts
rcParams['axes.linewidth'] = 1.2  # Border thickness
rcParams['axes.edgecolor'] = '#333333'  # Border color
rcParams['grid.color'] = '#D3D3D3'  # Grid color
rcParams['grid.alpha'] = 0.5  # Grid transparency

# Choose chart style
available_styles = plt.style.available
if 'seaborn-v0_8-darkgrid' in available_styles:
    plt.style.use('seaborn-v0_8-darkgrid')  # Prioritize this style
elif 'dark_background' in available_styles:
    plt.style.use('dark_background')  # Backup
else:
    plt.style.use('ggplot')  # Default

# Team palette (complete 20 teams)
TEAM_PALETTE = {
    'Liverpool': '#C8102E',
    'Arsenal': '#EF0107',
    'Manchester City': '#6CABDD',
    # ... (full palette remains)
    'Ipswich Town': '#3A70A3'
}
```

This section configures the visual appearance of the plots using rcParams to set fonts, line styles, grid colors, and other global parameters. It then attempts to apply a pre-defined plot style (like seaborn-v0_8-darkgrid) for a cohesive look. Finally, the TEAM_PALETTE dictionary assigns a specific color to each Premier League team for consistent and recognizable color-coding in all visualizations. This ensures clear and visually appealing plots throughout the analysis.

Most Common and General:

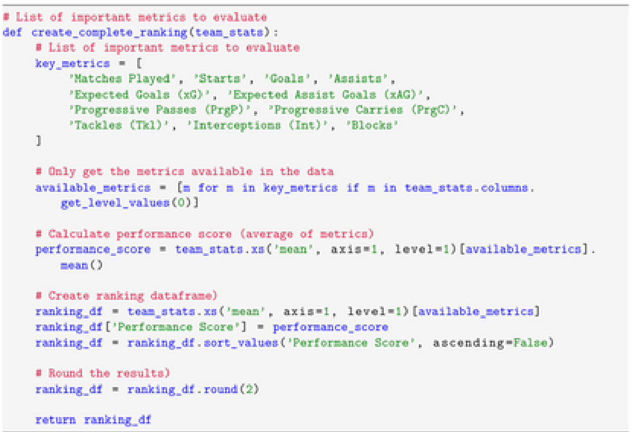


```python
def plot_enhanced_distribution(metric):
    # Set chart size in golden ratio
    golden_ratio = 1.618
    fig_width = 16
    fig_height = fig_width / golden_ratio

    # Create figure with high resolution
    plt.figure(figsize=(fig_width, fig_height), dpi=300)

    # Sort teams by average value of metric
    team_order = df.groupby('Team')[metric].mean().sort_values(ascending=False).index

    # Create gradient color for each team
    gradient_palette = {
        team: mcolors.LinearSegmentedColormap.from_list(
            f"team_gradient", ['#FFFFFF', color, color]
        ) for team, color in TEAM_PALETTE.items()
    }

    # Draw violin plot (data distribution)
    ax = sns.violinplot(
        data=df,
        x='Team',
        y=metric,
        order=team_order,
        palette=TEAM_PALETTE,
        inner=None,
        linewidth=1.5,
        saturation=0.8,
        alpha=0.9
    )

    # Add swarm plot (display each data point)
    sns.swarmplot(
        data=df,
        x='Team',
        y=metric,
        order=team_order,
        color='black',
        alpha=0.6,
```

Visualizes Data Distribution: Combines a violin plot (showing data density) and a swarm plot (displaying individual data points) to provide a comprehensive view of how a metric is distributed across different teams.
Sorts Teams by Performance: Orders the teams along the x-axis based on their average value for the selected metric, making it easy to identify top performers.
Highlights Top Teams: Visually emphasizes the top 3 teams by adding a gold border to their respective violin plots.
Displays Mean Values: Adds horizontal lines and text labels to show the mean value of the metric for each team, providing a clear point of reference.
Customizes Aesthetics: Uses the predefined TEAM_PALETTE for team-specific colors, sets a visually appealing plot size based on the golden ratio, and includes a title, labels, and a note for context.
Saves High-Quality Output: Saves the generated plot as a high-resolution PNG image to ensure clear and detailed visuals. In summary, it generates an informative, professional-looking visualization to compare team performance for a given metric.

PROBLEM 2

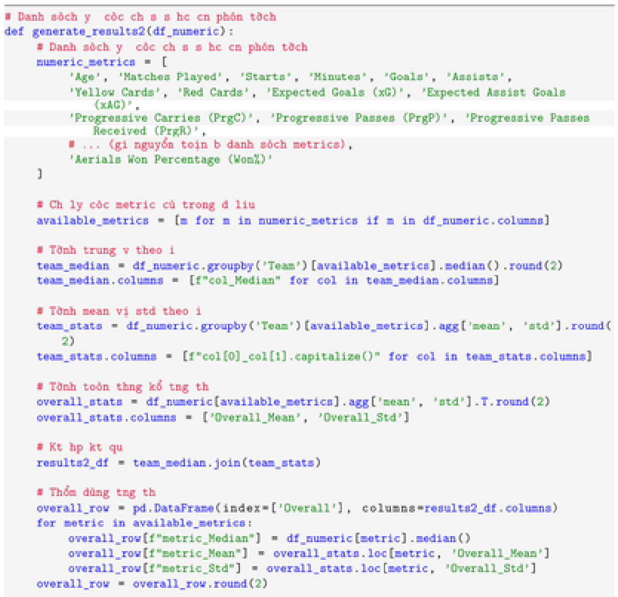
Complete Ranking Table:



```
# List of important metrics to evaluate
def create_complete_ranking(team_stats):
    # List of important metrics to evaluate
    key_metrics = [
        'Matches Played', 'Starts', 'Goals', 'Assists',
        'Expected Goals (xG)', 'Expected Assist Goals (xAG)',
        'Progressive Passes (PrgP)', 'Progressive Carries (PrgC)',
        'Tackles (Tkl)', 'Interceptions (Int)', 'Blocks'
    ]

    # Only get the metrics available in the data
    available_metrics = [m for m in key_metrics if m in team_stats.columns.
        get_level_values(0)]

    # Calculate performance score (average of metrics)
    performance_score = team_stats.xs('mean', axis=1, level=1)[available_metrics].
        mean()

    # Create ranking dataframe)
    ranking_df = team_stats.xs('mean', axis=1, level=1)[available_metrics]
    ranking_df['Performance Score'] = performance_score
    ranking_df = ranking_df.sort_values('Performance Score', ascending=False)

    # Round the results)
    ranking_df = ranking_df.round(2)

    return ranking_df
```

This function, create_complete_ranking, generates a team ranking based on key performance metrics. It calculates a performance score by averaging the specified metrics for each team and then sorts the teams accordingly. It returns the ranking as a Pandas DataFrame, rounded to two decimal places.

Complete results2.csv File Generation:



```
# Danh sóch y  cóc ch s s hc cn phón tðch
def generate_results2(df_numeric):
    # Danh sóch y  cóc ch s s hc cn phón tðch
    numeric_metrics = [
        'Age', 'Matches Played', 'Starts', 'Minutes', 'Goals', 'Assists',
        'Yellow Cards', 'Red Cards', 'Expected Goals (xG)', 'Expected Assist Goals
            (xAG)',
        'Progressive Carries (PrgC)', 'Progressive Passes (PrgP)', 'Progressive Passes
            Received (PrgR)',
        # ... (gi nguyển toịn b danh sóch metrics),
        'Aerials Won Percentage (Won%)'
    ]

    # Ch ly cóc metric cú trong d liu
    available_metrics = [m for m in numeric_metrics if m in df_numeric.columns]

    # Tðnh trung v theo i
    team_median = df_numeric.groupby('Team')[available_metrics].median().round(2)
    team_median.columns = [f"col_Median" for col in team_median.columns]

    # Tðnh mean vị std theo i
    team_stats = df_numeric.groupby('Team')[available_metrics].agg['mean', 'std'].round(
        2)
    team_stats.columns = [f"col[0]_col[1].capitalize()" for col in team_stats.columns]

    # Tðnh toón thng kổ tng th
    overall_stats = df_numeric[available_metrics].agg['mean', 'std'].T.round(2)
    overall_stats.columns = ['Overall_Mean', 'Overall_Std']

    # Kt hp kt qu
    results2_df = team_median.join(team_stats)

    # Thốm dùng tng th
    overall_row = pd.DataFrame(index=['Overall'], columns=results2_df.columns)
    for metric in available_metrics:
        overall_row[f"metric_Median"] = df_numeric[metric].median()
        overall_row[f"metric_Mean"] = overall_stats.loc[metric, 'Overall_Mean']
        overall_row[f"metric_Std"] = overall_stats.loc[metric, 'Overall_Std']
    overall_row = overall_row.round(2)
```

This function, generate_results2, creates the comprehensive results2.csv file. It calculates and compiles statistical summaries for each numeric metric in the dataset, including:

Team-Specific Statistics: Calculates the median, mean, and standard deviation for each metric, grouped by team. This provides insights into the distribution of each statistic within each team.

Overall Statistics: Calculates the overall mean and standard deviation for each metric across all players in the league, providing a league-wide benchmark.

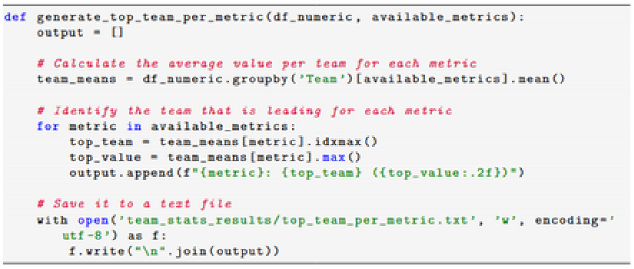
Data Aggregation: Combines the team-specific and overall statistics into a single Pandas DataFrame (results2_df).

Output to CSV: Exports the aggregated data to a results2.csv file, ensuring a well-structured format for further analysis. This includes a descriptive column naming convention and proper formatting of numeric values.

Available metrics: stores the available metrics in order to let generate the other file.

In short, this function efficiently generates a CSV file containing a wealth of statistical information about player performance, facilitating detailed comparisons between teams and the league as a whole.

Create top_team_per_metric.txt File:



```
def generate_top_team_per_metric(df_numeric, available_metrics):
    output = []

    # Calculate the average value per team for each metric
    team_means = df_numeric.groupby('Team')[available_metrics].mean()

    # Identify the team that is leading for each metric
    for metric in available_metrics:
        top_team = team_means[metric].idxmax()
        top_value = team_means[metric].max()
        output.append(f"{metric}: {top_team} ({top_value:.2f})")

    # Save it to a text file
    with open('team_stats_results/top_team_per_metric.txt', 'w', encoding='
        utf-8') as f:
        f.write("\n".join(output))
```

This function, generate_top_team_per_metric, generates a text file (top_team_per_metric.txt) that identifies the leading team for each available metric.

Calculates Team Averages: It first calculates the average value of each metric for every team using the groupby() and mean() functions.

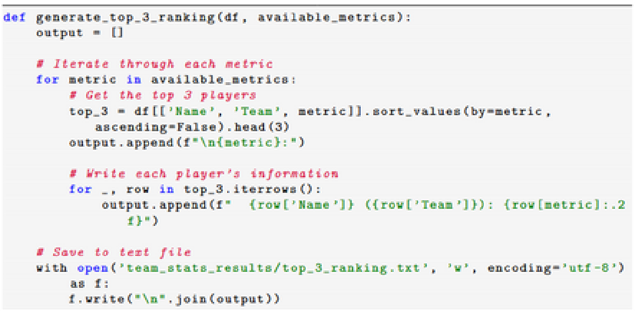
Identifies Top Teams: For each metric, it identifies the team with the highest average value using idxmax() and retrieves the corresponding maximum value using max().

Formats Output: It then formats the output as a string including the metric name, the team name, and the team's average value, rounded to two decimal places.

Saves Results to File: Finally, it writes these results to the top_team_per_metric.txt file, providing a concise summary of which team excels in each category.

In essence, this function quickly summarizes which team performs best for each statistic being considered.

Create top_3_ranking.txt File:



```
def generate_top_3_ranking(df, available_metrics):
    output = []

    # Iterate through each metric
    for metric in available_metrics:
        # Get the top 3 players
        top_3 = df[['Name', 'Team', metric]].sort_values(by=metric,
            ascending=False).head(3)
        output.append(f"\n{metric}:")

        # Write each player's information
        for _, row in top_3.iterrows():
            output.append(f"  {row['Name']} ({row['Team']}): {row[metric]:.2
                f}")

    # Save to text file
    with open('team_stats_results/top_3_ranking.txt', 'w', encoding='utf-8')
        as f:
        f.write("\n".join(output))
```

This function, generate_top_3_ranking, creates a text file (top_3_ranking.txt) containing the top 3 players for each available metric.

Iterates Through Metrics: The code loops through each metric in the available_metrics list.

Extracts Top 3 Players: For each metric, it selects the player name, team, and the metric itself. Then, it sorts the players based on the metric's value in descending order and extracts the top 3 players using .head(3).

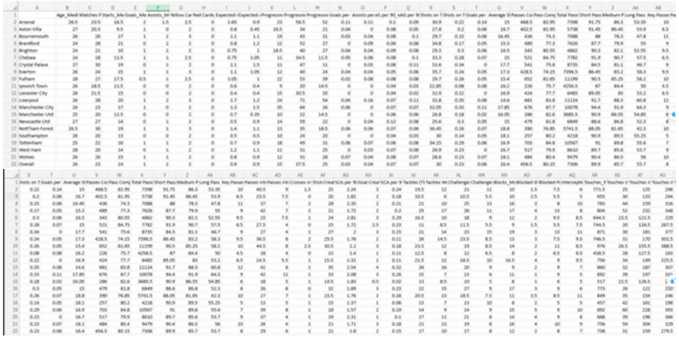
Formats Output: It formats each player's information (name, team, and metric value) into a string.

Saves to File: Finally, it saves the formatted output to the top_3_ranking.txt file, providing a quick reference to the best players in each statistical category.

6. Results

Calculate team statistics:

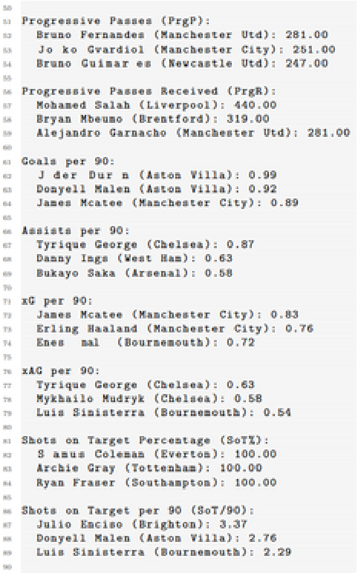
The output is results2.csv



PROBLEM 2

Top 3 player information

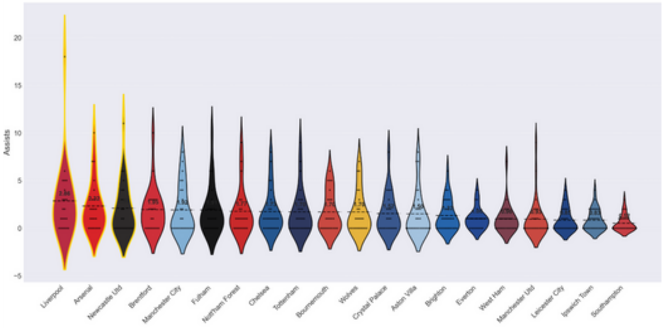
The output is top_3_ranking.txt



```

Progressive Passes (PrgP):
  Bruno Fernandes (Manchester Utd): 281.00
   Jo ko Gvardiol (Manchester City): 251.00
  Bruno Guimar es (Newcastle Utd): 247.00

Progressive Passes Received (PrgR):
  Mohamed Salah (Liverpool): 440.00
  Bryan Mbeumo (Brentford): 319.00
  Alejandro Garnacho (Manchester Utd): 281.00

Goals per 90:
   J der Dur n (Aston Villa): 0.99
  Donyell Malen (Aston Villa): 0.92
  James Mcatee (Manchester City): 0.89

Assists per 90:
  Tyrique George (Chelsea): 0.87
  Danny Ings (West Ham): 0.63
  Bukayo Saka (Arsenal): 0.58

xG per 90:
  James Mcatee (Manchester City): 0.83
  Erling Haaland (Manchester City): 0.76
  Enes  mal  (Bournemouth): 0.72

xAG per 90:
  Tyrique George (Chelsea): 0.63
  Mykhailo Mudryk (Chelsea): 0.58
  Luis Sinisterra (Bournemouth): 0.54

Shots on Target Percentage (SoT%):
  S amus Coleman (Everton): 100.00
  Archie Gray (Tottenham): 100.00
  Ryan Fraser (Southampton): 100.00

Shots on Target per 90 (SoT/90):
  Julio Enciso (Brighton): 3.37
  Donyell Malen (Aston Villa): 2.76
  Luis Sinisterra (Bournemouth): 2.29

```

Histogram

Distribution of Assists by Team (2024-2025 Season)



Explanation of Chart Elements:

PrgP Distribution: Each "teardrop" shape represents the distribution of the number of progressive passes each team made in their matches.

Horizontal Line: This line typically represents the mean or median number of progressive passes the team makes per match.

Dots: Each dot represents the specific number of progressive passes the team made in an individual match.

Team Order: The teams are likely arranged in order from the team with the highest to the lowest average number of progressive passes.

Chart Interpretation:

Playing Style: This chart shows which teams tend to make more progressive passes. Progressive passes move the ball closer to the opponent's goal, so this chart reflects the team's attacking style and tendency to move the ball forward.

Consistency: The width of the "teardrop" shows consistency. A narrow "teardrop" indicates the team has a fairly stable number of progressive passes across matches, while a wide "teardrop" shows greater variability.

Match Comparison: Individual dots show the difference in the number of progressive passes in each match.

Example:

Manchester Utd has the highest "teardrop" and the highest horizontal line, which shows they make more progressive passes than other teams in the league. This may suggest they tend to play offensively and move the ball forward quickly.

Liverpool has a relatively wide "teardrop," indicating they tend to make progressive passes fairly consistently compared to other teams.

Distribution of Expected Goals (xG) by Team (2024-2025 Season)

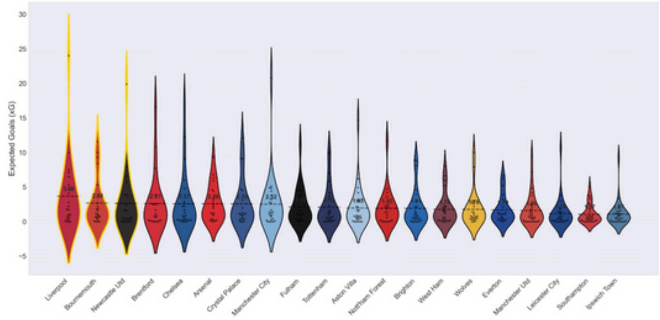


Chart Information:

xG Distribution: Each "teardrop" represents the distribution of the number of Expected Goals (xG) achieved by a specific team. The width of the teardrop at a particular xG level indicates the number of matches where the team had an xG close to that level.

Dashed Line: The dashed line within each teardrop typically indicates the mean or median xG for that team.

Points: Small dots may represent the actual xG in individual matches.

Team Order: The teams are arranged in a specific order, likely based on their average or total xG.

Data Source: "Premier League 2024-2025" indicates the data source and the season on which the chart is based.

Chart Interpretation:

Offensive Performance Evaluation: The chart allows you to compare the offensive performance of different teams. Teams with higher teardrops and dashed lines positioned higher have generated more scoring opportunities (based on xG).

Consistency: The shape of the teardrop can indicate a team's consistency. A narrower teardrop suggests the team has a relatively consistent xG in each match, while a wider teardrop suggests greater variability.

Individual Match Comparison: The points can provide insights into how xG differs between matches.

Example:

"Liverpool" has a tall teardrop with a high dashed line, meaning they are leading the league in xG.

Goals Distribution by Team (2024-2025 Season)

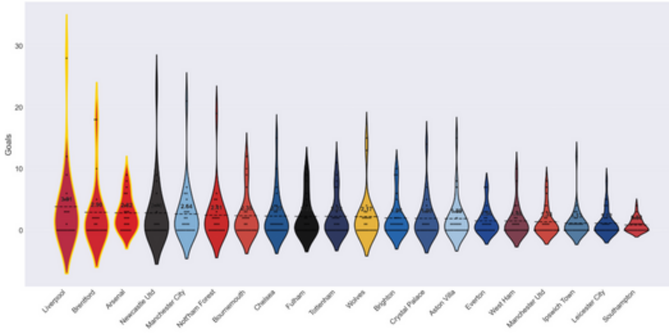


Chart Information:

Primary Purpose: This chart compares the goal-scoring abilities of teams in the league.

"Teardrop" Shape: Shows the distribution of the number of goals scored in each match by a team. A tall, narrow "teardrop" indicates consistent goal-scoring, while a wide "teardrop" suggests greater variability.

Horizontal Line: Represents the mean or median number of goals scored by the team per match.

Team Placement: Teams are arranged in order of average/total goals.

Example:

Liverpool has the tallest "teardrop," which means they are the highest-scoring team. At the same time, the dots indicate that they sometimes score a lot of goals in a single match.

Southampton has a lower "teardrop," so they score fewer goals than Liverpool.

PROBLEM 2

Distribution of Progressive Passes (PrgP) by Team (2024-2025 Season)

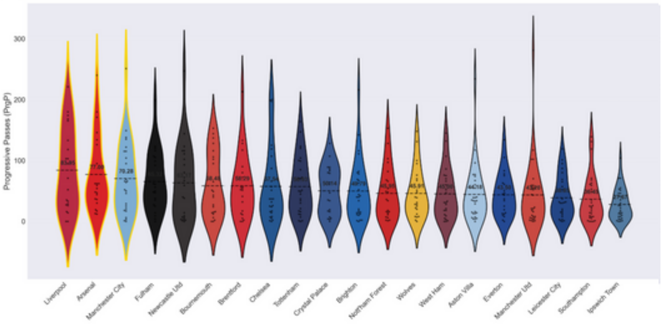


Chart Information:
Assist Distribution: Each "teardrop" represents the distribution of the number of assists made by a specific team. The width of the teardrop at a particular assist count indicates the number of matches where the team had an assist count close to that level.
Dashed Line: The dashed line within each teardrop typically indicates the mean or median number of assists for that team.
Points: Small dots may represent the actual number of assists in individual matches.
Team Order: The teams are arranged in a specific order, likely based on their average or total number of assists.
Data Source: "Premier League 2024-2025" indicates the data source and the season on which the chart is based.
Chart Interpretation:
Team Performance Evaluation: The chart allows you to compare the offensive performance of different teams. Teams with higher teardrops and dashed lines positioned higher have generated more assists.
Consistency: The shape of the teardrop can indicate a team's consistency. A narrower teardrop suggests the team has a relatively consistent number of assists in each match, while a wider teardrop suggests greater variability.
Individual Match Comparison: The points can provide insights into how the number of assists varies between matches.
Example:
"**Liverpool**" has a tall, red teardrop with a high dashed line, meaning they are leading the league in assists.

Conclusion

Based on the statistical data, the analysis predicts that **Liverpool** will be the champion of the 2024-2025 season.

PROBLEM 3

1.Objectives

Cluster players based on key statistical performance indicators (such as Goals per 90, Assists per 90, xG per 90, Tackles Won, etc.).
Determine the optimal number of clusters using the Elbow method combined with evaluation metrics (Silhouette Score, Davies-Bouldin Score).
Apply **UMAP** (instead of PCA) for dimensionality reduction to improve clustering quality and facilitate visualization.
Analyze the characteristics of each cluster to understand the strengths and weaknesses of each player group.

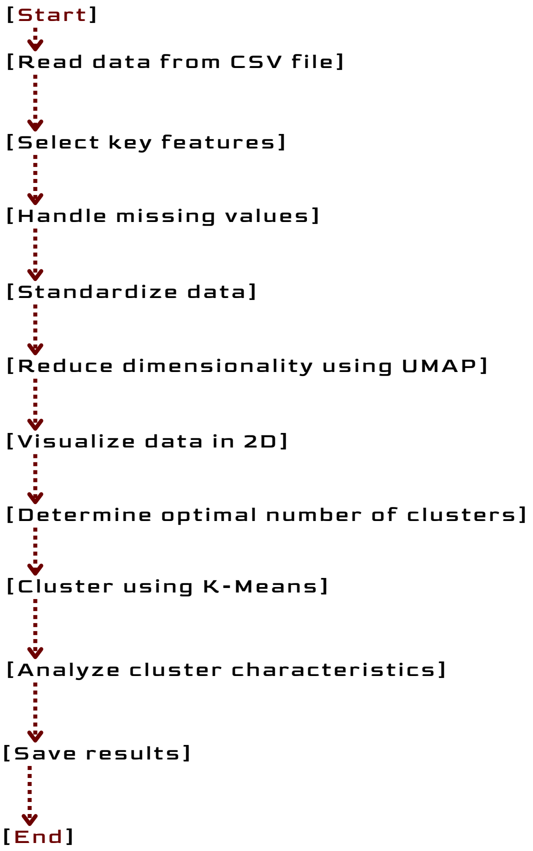
2. Methodology

Data Preprocessing
Feature Selection: Select 31 key statistical indicators related to player performance.
Missing Value Handling: Impute missing values by replacing them with the median of each respective column.
Data Standardization: Apply StandardScaler to ensure all features are on the same scale.
Dimensionality Reduction (UMAP)
Why UMAP instead of PCA?
UMAP preserves both local and global data structure better than PCA, leading to more effective clustering.
UMAP handles non-linear data well (PCA is only suitable for linear data).
UMAP allows tuning of n_neighbors and min_dist to control cluster compactness.
Clustering with K-Means
Determining the Optimal Number of Clusters:
Use the Silhouette Score (higher is better) and the Davies-Bouldin Score (lower is better) to evaluate cluster quality.
Experiment with cluster counts from 2 to 10, selecting the optimal value based on the Elbow method plot.
Cluster Analysis:
Calculate the centroid of each cluster.
Identify prominent features (top_features) and representative players (closest to the centroid).

3.Source Code

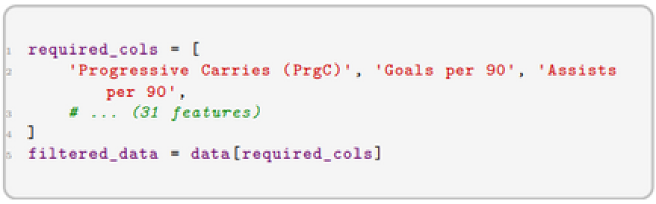
https://github.com/duahauhz/ASSIGNMENT_01_PYTHON_PTIT/blob/master/RESULTS/TASK%203/assignment_01_task_03.py

4.Code Execution Flowchart



5. Explain The Code

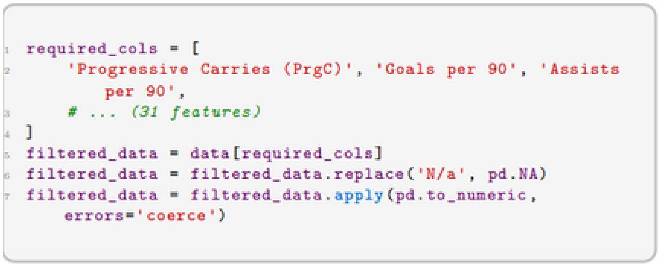
Feature Selection:



```python
required_cols = [
    'Progressive Carries (PrgC)', 'Goals per 90', 'Assists per 90',
    # ... (31 features)
]
filtered_data = data[required_cols]
```

Select 31 key statistical indicators for the analysis.
filtered_data now contains only the selected columns.

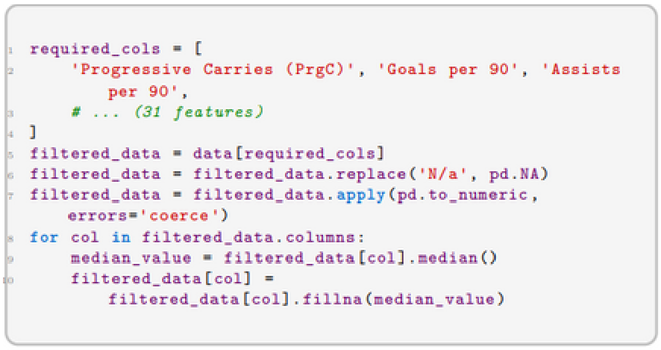
Missing Value Handling:



```python
required_cols = [
    'Progressive Carries (PrgC)', 'Goals per 90', 'Assists per 90',
    # ... (31 features)
]
filtered_data = data[required_cols]
filtered_data = filtered_data.replace('N/a', pd.NA)
filtered_data = filtered_data.apply(pd.to_numeric,
    errors='coerce')
```

Replace 'N/a' values with NaN (Not a Number).

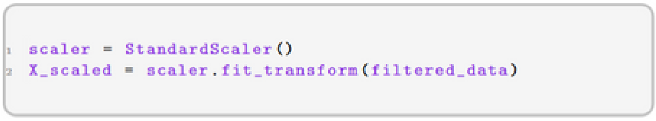
Convert all data to numeric type; non-numeric values will be coerced to NaN.



```python
required_cols = [
    'Progressive Carries (PrgC)', 'Goals per 90', 'Assists per 90',
    # ... (31 features)
]
filtered_data = data[required_cols]
filtered_data = filtered_data.replace('N/a', pd.NA)
filtered_data = filtered_data.apply(pd.to_numeric,
    errors='coerce')
for col in filtered_data.columns:
    median_value = filtered_data[col].median()
    filtered_data[col] =
        filtered_data[col].fillna(median_value)
```

Impute missing values with the median of each column to mitigate the impact of outliers.

Data Standardization:

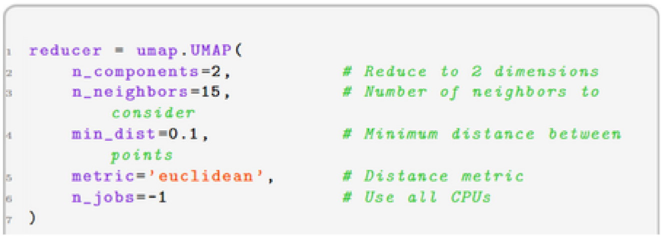


```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(filtered_data)
```

Standardize the data to a common scale (mean=0, std=1) to prevent features with large values from dominating the algorithm.
X_scaled is the standardized data matrix.

Dimensionality Reduction with UMAP:

UMAP Initialization

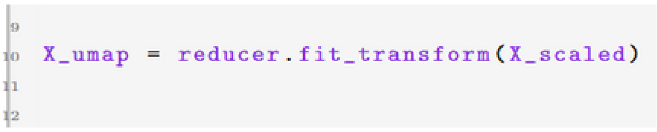


```python
reducer = umap.UMAP(
    n_components=2,          # Reduce to 2 dimensions
    n_neighbors=15,          # Number of neighbors to consider
    min_dist=0.1,            # Minimum distance between points
    metric='euclidean',      # Distance metric
    n_jobs=-1                # Use all CPUs
)
```

PROBLEM 3

n_neighbors=15: Balances local and global data structure preservation.
min_dist=0.1: Controls the compactness of points after dimensionality reduction.

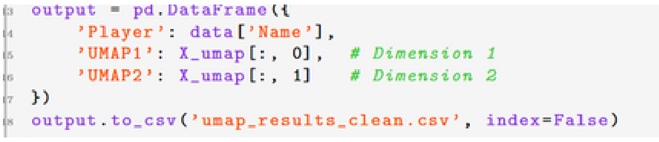
Perform Dimensionality Reduction:



```python

X_umap = reducer.fit_transform(X_scaled)


```

X_umap is the 2D matrix (2 columns) representing the original data.

Save Dimensionality Reduction Results:

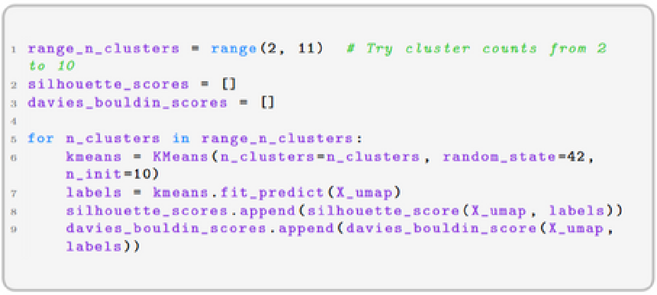


```python
output = pd.DataFrame({
    'Player': data['Name'],
    'UMAP1': X_umap[:, 0],   # Dimension 1
    'UMAP2': X_umap[:, 1]    # Dimension 2
})
output.to_csv('umap_results_clean.csv', index=False)
```

Export to a CSV file for later analysis.

Determining the Optimal Number of Clusters:

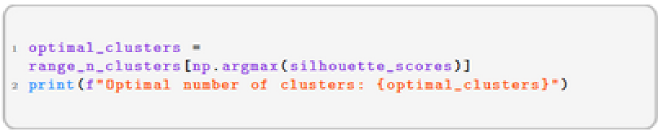
Calculating Evaluation Metrics:



```python
range_n_clusters = range(2, 11)  # Try cluster counts from 2 to 10
silhouette_scores = []
davies_bouldin_scores = []

for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_umap)
    silhouette_scores.append(silhouette_score(X_umap, labels))
    davies_bouldin_scores.append(davies_bouldin_score(X_umap, labels))
```

Silhouette Score: Measures the separation between clusters (higher is better).
Davies-Bouldin Score: Measures the distance between clusters (lower is better).

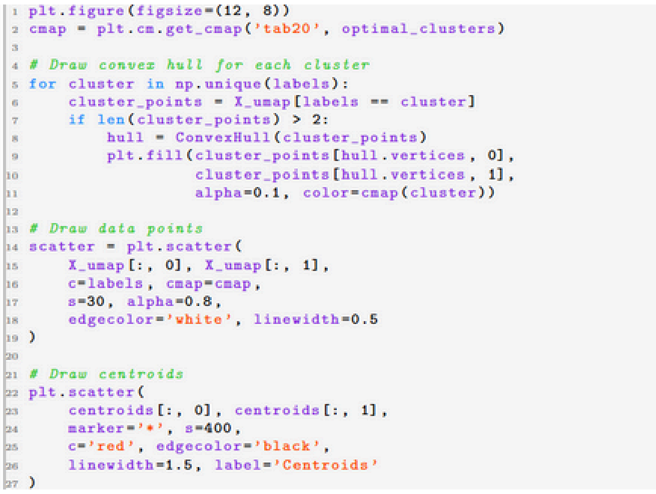
Selecting the Optimal Number of Clusters:



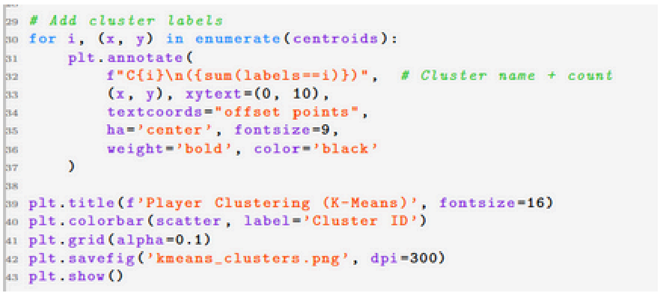
```python
optimal_clusters = range_n_clusters[np.argmax(silhouette_scores)]
print(f"Optimal number of clusters: {optimal_clusters}")
```

Select the number of clusters with the highest Silhouette Score.

Visualizing Results:



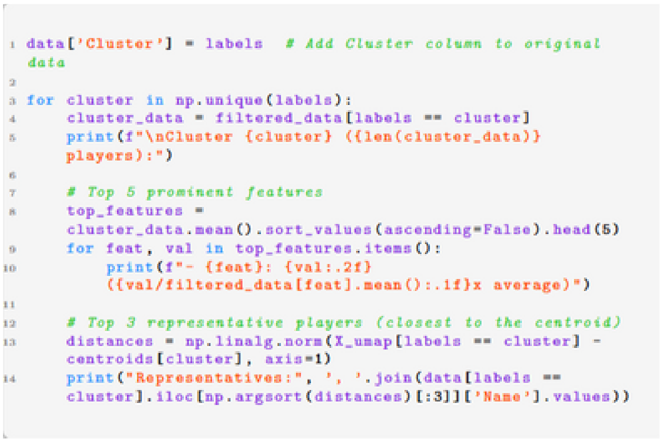
```python
plt.figure(figsize=(12, 8))
cmap = plt.cm.get_cmap('tab20', optimal_clusters)

# Draw convex hull for each cluster
for cluster in np.unique(labels):
    cluster_points = X_umap[labels == cluster]
    if len(cluster_points) > 2:
        hull = ConvexHull(cluster_points)
        plt.fill(cluster_points[hull.vertices, 0],
                 cluster_points[hull.vertices, 1],
                 alpha=0.1, color=cmap(cluster))

# Draw data points
scatter = plt.scatter(
    X_umap[:, 0], X_umap[:, 1],
    c=labels, cmap=cmap,
    s=30, alpha=0.8,
    edgecolor='white', linewidth=0.5
)

# Draw centroids
plt.scatter(
    centroids[:, 0], centroids[:, 1],
    marker='*', s=400,
    c='red', edgecolor='black',
    linewidth=1.5, label='Centroids'
)
```



```python
# Add cluster labels
for i, (x, y) in enumerate(centroids):
    plt.annotate(
        f"C{i}\n({sum(labels==i)})",  # Cluster name + count
        (x, y), xytext=(0, 10),
        textcoords="offset points",
        ha='center', fontsize=9,
        weight='bold', color='black'
    )

plt.title(f'Player Clustering (K-Means)', fontsize=16)
plt.colorbar(scatter, label='Cluster ID')
plt.grid(alpha=0.1)
plt.savefig('kmeans_clusters.png', dpi=300)
plt.show()
```

Convex Hull helps enclose each cluster for easy visualization.
Centroids are marked with red stars.
Each cluster has its own color and displays the number of players.

Analyzing Cluster Characteristics:

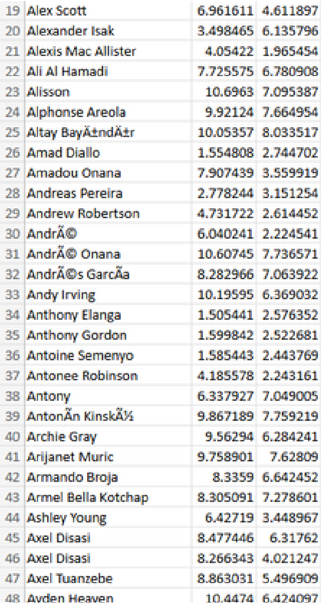


```python
data['Cluster'] = labels  # Add Cluster column to original data

for cluster in np.unique(labels):
    cluster_data = filtered_data[labels == cluster]
    print(f"\nCluster {cluster} ({len(cluster_data)} players):")

    # Top 5 prominent features
    top_features = cluster_data.mean().sort_values(ascending=False).head(5)
    for feat, val in top_features.items():
        print(f"- {feat}: {val:.2f} ({val/filtered_data[feat].mean():.1f}x average)")

    # Top 3 representative players (closest to the centroid)
    distances = np.linalg.norm(X_umap[labels == cluster] - centroids[cluster], axis=1)
    print("Representatives:", ', '.join(data[labels == cluster].iloc[np.argsort(distances)[:3]]['Name'].values))
```

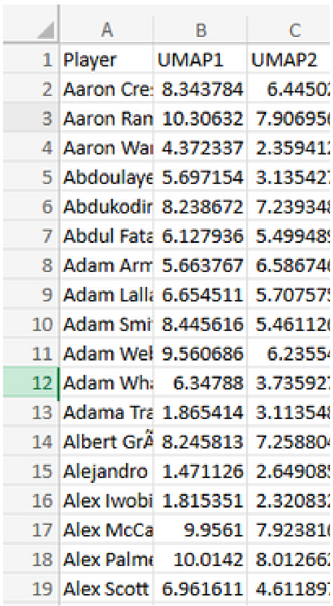
6. Results

UMAP Dimensionality Reduction Results:

umap_results_clean.csv



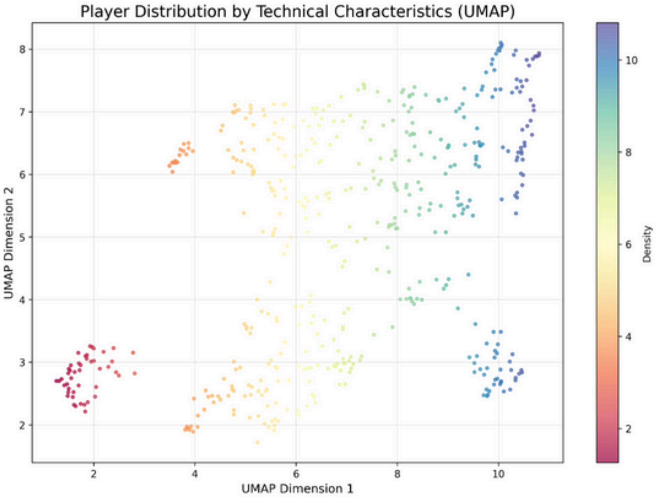
19	Alex Scott	6.961611	4.611897
20	Alexander Isak	3.498465	6.135796
21	Alexis Mac Allister	4.05422	1.965454
22	Ali Al Hamadi	7.725575	6.780908
23	Alisson	10.6963	7.095387
24	Alphonse Areola	9.92124	7.664954
25	Altay BayÄ±ndÄ±r	10.05357	8.033517
26	Amad Diallo	1.554808	2.744702
27	Amadou Onana	7.907439	3.559919
28	Andreas Pereira	2.778244	3.151254
29	Andrew Robertson	4.731722	2.614452
30	AndrÃ©	6.040241	2.224541
31	AndrÃ© Onana	10.60745	7.736571
32	AndrÃ©s GarcÃa	8.282966	7.063922
33	Andy Irving	10.19595	6.369032
34	Anthony Elanga	1.505441	2.576352
35	Anthony Gordon	1.599842	2.522681
36	Antoine Semenyo	1.585443	2.443769
37	Antonee Robinson	4.185578	2.243161
38	Antony	6.337927	7.049005
39	AntonÃn KinskÃ½	9.867189	7.759219
40	Archie Gray	9.56294	6.284241
41	Arijanet Muric	9.758901	7.62809
42	Armando Broja	8.3359	6.642452
43	Armel Bella Kotchap	8.305091	7.278601
44	Ashley Young	6.42719	3.448967
45	Axel Disasi	8.477446	6.31762
46	Axel Disasi	8.266343	4.021247
47	Axel Tuanzebe	8.863031	5.496909
48	Ayden Heaven	10.4474	6.424097



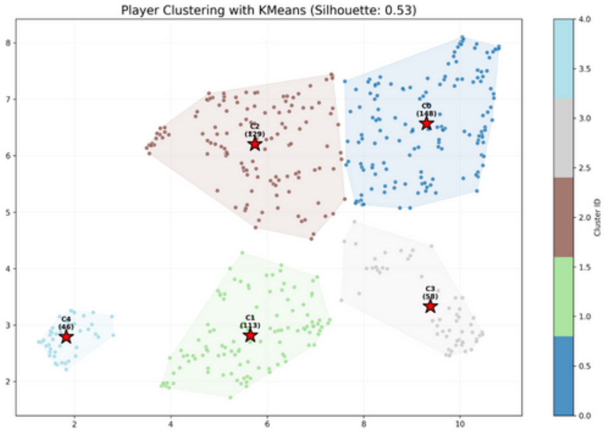
	A	B	C
1	Player	UMAP1	UMAP2
2	Aaron Cre:	8.343784	6.4450:
3	Aaron Ran	10.30632	7.906951
4	Aaron Wa	4.372337	2.35941:
5	Abdoulaye	5.697154	3.13542
6	Abdukodir	8.238672	7.23934
7	Abdul Fat:	6.127936	5.49948
8	Adam Arr	5.663767	6.58674
9	Adam Lalli	6.654511	5.70757
10	Adam Smi	8.445616	5.46112
11	Adam Wel	9.560686	6.2355
12	Adam Wh:	6.34788	3.73592
13	Adama Tra	1.865414	3.11354
14	Albert GrÃ	8.245813	7.25880
15	Alejandro	1.471126	2.64908
16	Alex Iwobi	1.815351	2.32083
17	Alex McCa	9.9561	7.92381
18	Alex Palm	10.0142	8.01266
19	Alex Scott	6.961611	4.61189

PROBLEM 3

Data Visualization:

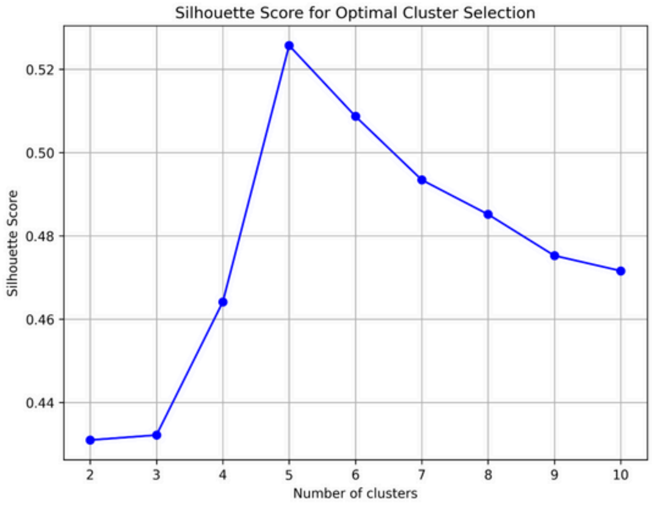


Results After Clustering:

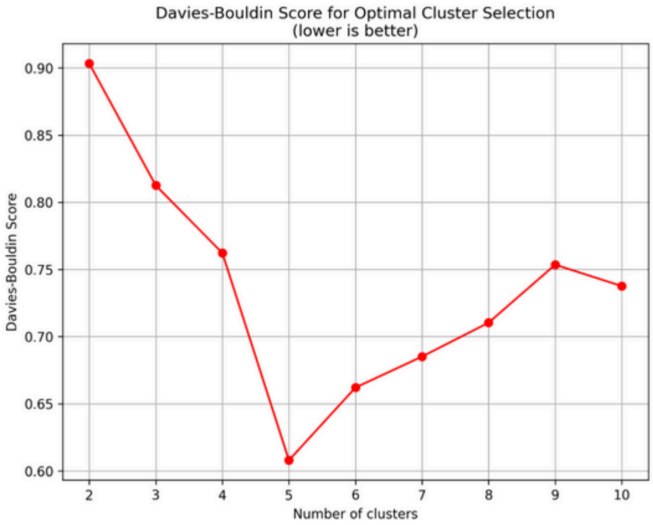


Visualizing Cluster Selection:

Silhouette Scores:



Davies-Bouldin:



Based on the two charts, we can see that the data can be divided into 5 clusters.

PROBLEM 4

1.Objectives

The primary objectives of this project are:

Collect Premier League 2024-2025 player data from footballtransfers.com, including only players with greater than 900 minutes of playing time.

Develop a model to predict transfer values based on performance features.

Evaluate and deploy the best model for predicting player value.

Rationale for this approach:

Focusing on players with sufficient playing time (>900 minutes) ensures data reliability.

Employing multiple models (Random Forest, XGBoost, etc.) to identify the optimal prediction model.

Combining both statistical data and expert ratings.

2. Methodology

The project implementation comprises four main steps:

Data Collection:

Use Selenium + BeautifulSoup to scrape data from footballtransfers.com

Filter for players with >900 minutes of playing time.

Save data to CSV.

Preprocessing:

Handle missing values and standardize data.

Convert categorical features (Position, Nation, etc.) using one-hot encoding.

Scale numerical features using StandardScaler.

Model Building:

Experiment with 3 models: Random Forest, Gradient Boosting, XGBoost.

Evaluate using MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error).

Optimize hyperparameters using GridSearchCV.

Evaluation and Deployment:

Analyze feature importance.

Save the best performing model.

Predict values for all players.

Rationale for this methodology:

Selenium is well-suited for dynamic websites requiring JavaScript interaction.

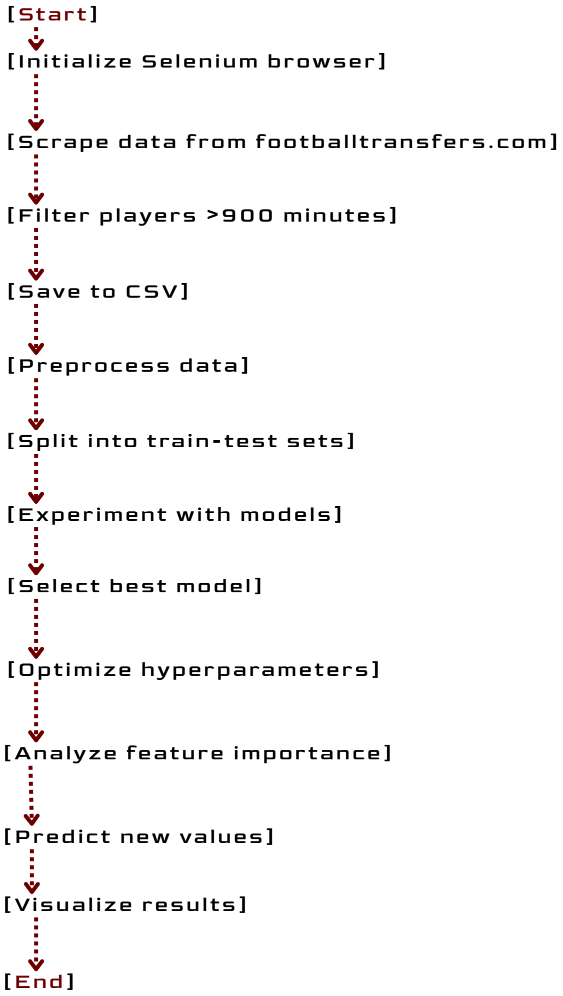
XGBoost often performs well with datasets containing both numerical and categorical features.

Data standardization helps models converge more effectively.

3.Source Code

https://github.com/duahauhz/ASSIGNMENT_01_PYTHON_PTIT/blob/master/RESULTS/TASK%204/assignment_01_task_04.ipynb

4.Code Execution Flowchart



[Start]

[Initialize Selenium browser]

[Scrape data from footballtransfers.com]

[Filter players >900 minutes]

[Save to CSV]

[Preprocess data]

[Split into train-test sets]

[Experiment with models]

[Select best model]

[Optimize hyperparameters]

[Analyze feature importance]

[Predict new values]

[Visualize results]

[End]

5. Explain The Code

1. Scraping Module (scrape_data.py):

initialize_browser():

Purpose: Initializes a headless Chrome browser for scraping data.

Operation: Sets up options to run without a graphical interface, saving system resources.

extract_page_data():

Purpose: Extracts player information from the HTML of a specific webpage.

Operation: Uses BeautifulSoup to parse the HTML, searches for elements containing player information (name, team, market value, rating), and processes special values (such as ratings in the "X/Y" format).

main() (Scraping):

Purpose: Coordinates the data scraping process across multiple pages.

Operation: Iterates through the pages, calls extract_page_data() for each page, and saves the results to a CSV file.

2. Data Preprocessing Module (preprocessing.py):

clean_numeric():

Purpose: Standardizes numerical values (handling special characters, missing values).

Operation: Removes non-numeric characters (€, M, %), processes fractional values (X/Y), and converts to float type.

preprocess_data():

Purpose: Prepares the data for model training.

Operation: Handles missing values, encodes categorical features (one-hot encoding), and standardizes numerical data using StandardScaler.

PROBLEM 4

3. Modeling Module (model_training.py):

GridSearchCV():

Purpose: Searches for the optimal hyperparameters for the machine learning model.

Operation: Experiments with different combinations of hyperparameters and evaluates performance using cross-validation.

fit() (Models):

Purpose: Trains the model on the dataset.

Operation: Applies a machine learning algorithm (Random Forest, XGBoost, etc.) to learn patterns from the data.

predict():

Purpose: Predicts player values based on input features.

Operation: Uses the trained model to make predictions.

4. Prediction & Evaluation Module:

mean_absolute_error() / mean_squared_error():

Purpose: Evaluates the accuracy of the model.

Operation: Calculates the difference between predicted values and actual values.

feature_importances_:

Purpose: Identifies the most important features affecting player value.

Operation: Analyzes the contribution of each feature in the model.

5. Visualization Module:

sns.histplot() / plt.barplot():

Purpose: Visualizes value distributions and top players.

Operation: Draws charts from the processed data.

6. Utility Functions:

joblib.dump() / joblib.load():

Purpose: Saves and loads the model for reuse.

Operation: Serializes the model to a .pkl file and loads it when needed.

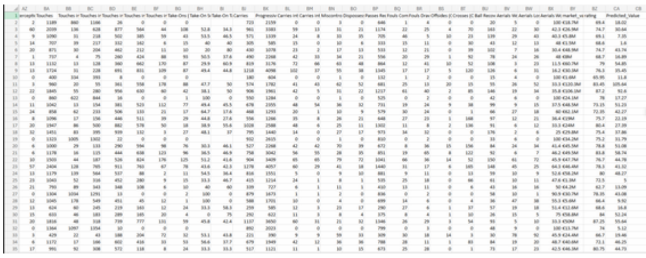
pd.get_dummies():

Purpose: Encodes categorical features into numerical format.

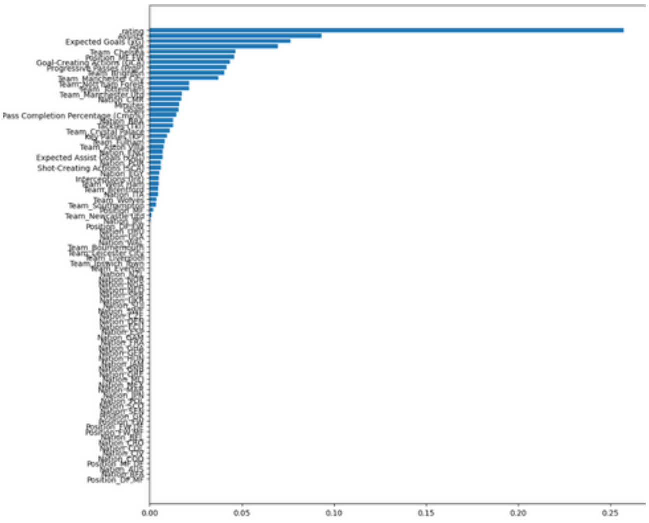
Operation: Creates binary columns corresponding to each categorical value.

6. Results

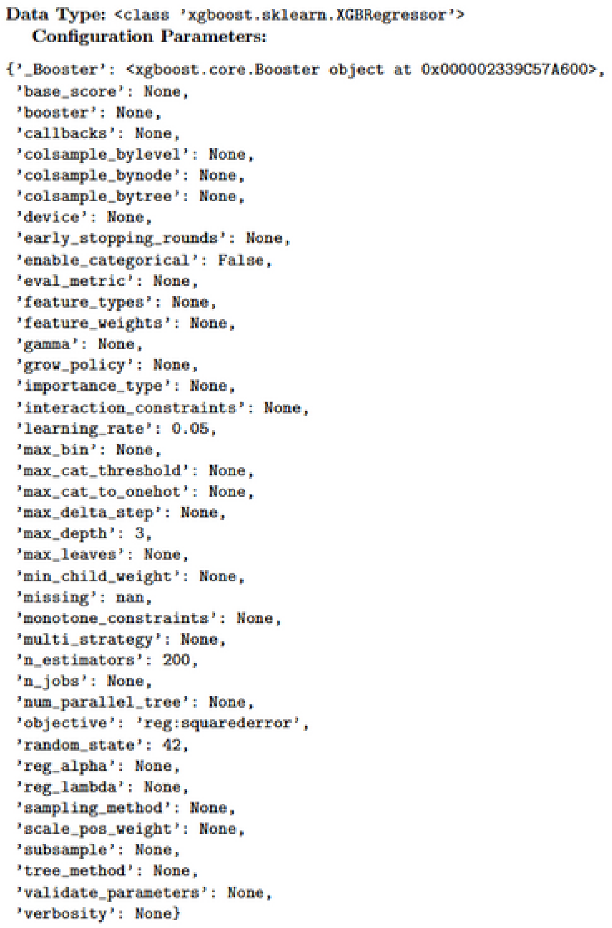
Output of Processed Data: The processed player data is outputted to the file all_players_predictions.csv:



Analyze the importance of each feature.



Output of Processed Data: The processed player data is outputted to the file all_players_predictions.csv:

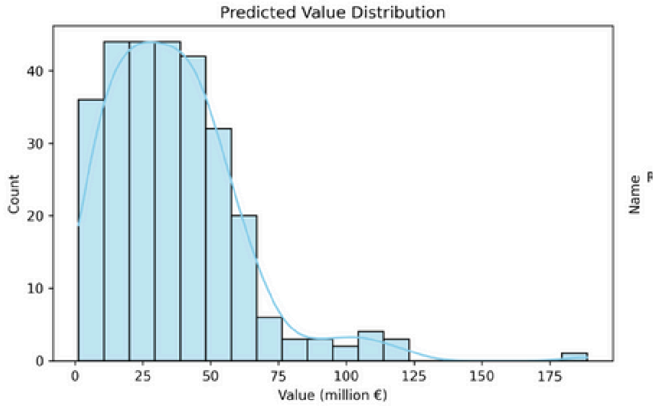


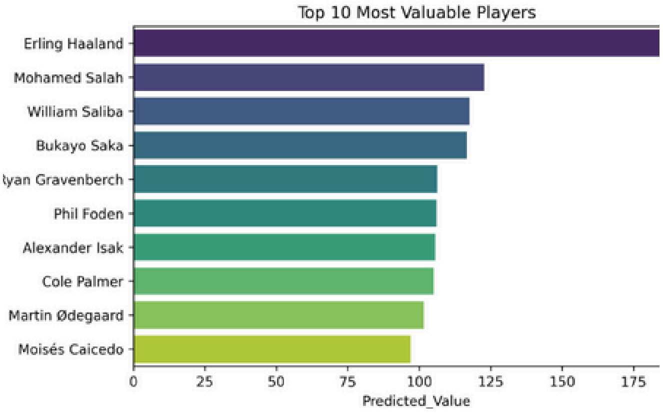
Data Type: `<class 'xgboost.sklearn.XGBRegressor'>`

Configuration Parameters:

```
{'_Booster': <xgboost.core.Booster object at 0x000002339C57A600>,
 'base_score': None,
 'booster': None,
 'callbacks': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'device': None,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'feature_types': None,
 'feature_weights': None,
 'gamma': None,
 'grow_policy': None,
 'importance_type': None,
 'interaction_constraints': None,
 'learning_rate': 0.05,
 'max_bin': None,
 'max_cat_threshold': None,
 'max_cat_to_onehot': None,
 'max_delta_step': None,
 'max_depth': 3,
 'max_leaves': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'multi_strategy': None,
 'n_estimators': 200,
 'n_jobs': None,
 'num_parallel_tree': None,
 'objective': 'reg:squarederror',
 'random_state': 42,
 'reg_alpha': None,
 'reg_lambda': None,
 'sampling_method': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'validate_parameters': None,
 'verbosity': None}
```

Analyze feature importance scores for each indicator:





PROBLEM 4

Distribution of Predicted Values

Shape and Range:

The histogram shows a right-skewed distribution, with the majority of predicted values ranging from 0 to 50 million €.

The highest peak is around 20–30 million €, indicating that most players are predicted to have an average to decent value.

There is a long tail extending to 175 million €, suggesting a few players have very high predicted values.

The KDE line smooths the distribution, reinforces the skew, and emphasizes the rarity of highly valued players.

Comments:

The right skew suggests that the dataset likely contains many average to lower-tier players, with a small elite group driving the high end of the value spectrum. This is typical in football, where a small elite group (e.g., superstars) have significantly higher market values.

The presence of values up to 175 million € suggests the model can identify top talents, aligning with real-world transfer market trends (e.g., transfer fees for players like Erling Haaland or Kylian Mbappé).

Significance:

The model may be influenced by features strongly correlated with player value, such as performance statistics (e.g., goals, assists), age, position, or team quality.

The long tail could indicate outliers or a need to investigate whether the model is overvaluing a few players.

Top 10 Most Valuable Players

Ranking and Values:

The bar chart lists the top 10 players by predicted value, with Erling Haaland leading at approximately 175 million €, followed by Mohamed Salah at approximately 160 million €, and other players like William Saliba, Bukayo Saka, and Ryan Gravenberch ranging from 100 to 150 million €.

The values gradually decrease, with Moisés Caicedo at the bottom of the top 10, around 75–100 million €.

Comments:

The top players (Haaland, Salah) are globally recognized stars, indicating the model effectively captures factors like goal-scoring ability, reputation, and market demand, which are key drivers of player value.

The appearance of young players like Saliba, Saka, and Caicedo (emerging talents) suggests the model can consider potential or future performance, possibly influenced by the rating column (average of current and potential rating) or features like age.

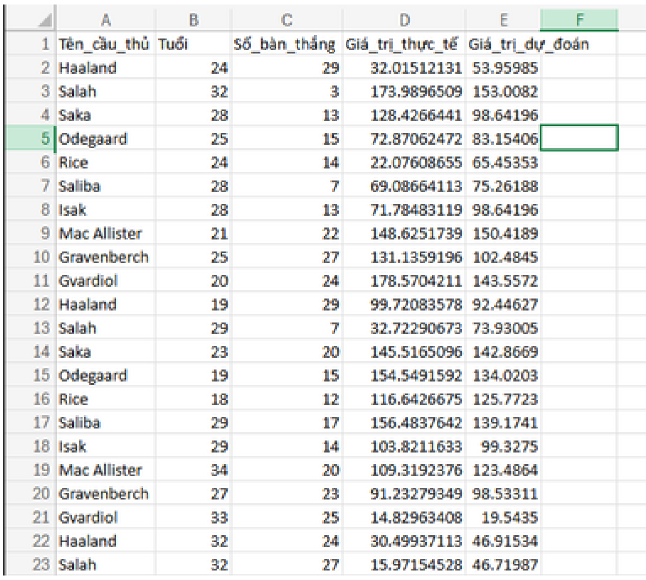
The high and stable values (75–175 million €) for the top 10 reflect a realistic hierarchy, with slight differences potentially due to team, position, or recent performance.

Significance:

The model aligns with the real-world football economy, where top strikers (e.g., Haaland) and versatile attacking players (e.g., Salah, Saka) command the highest prices.

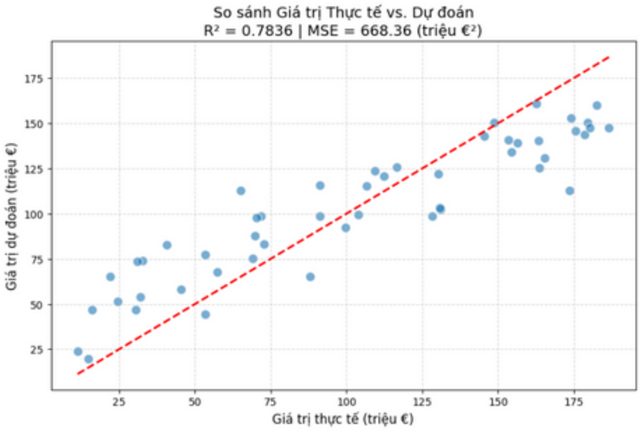
The presence of midfielders (e.g., Gravenberch, Caicedo) and defenders (e.g., Saliba) in the top 10 indicates the model values positional versatility or defensive contributions, which may be encoded in features like Position or team success metrics.

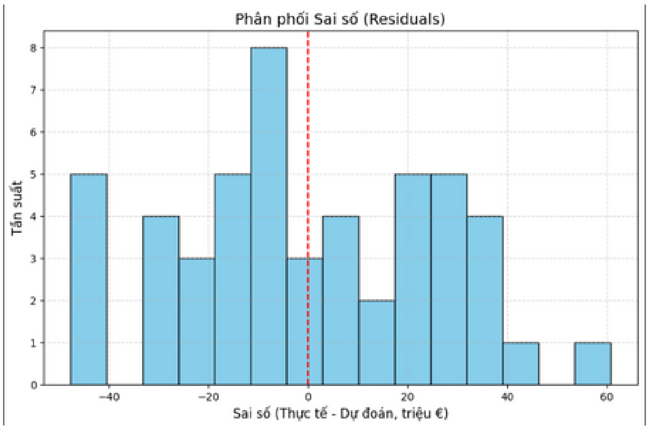
Player Transfer Value Predictions: player_value_predictions.csv



	A	B	C	D	E	F
1	Tên_cầu_thủ	Tuổi	Số_bàn_thắng	Giá_trị_thực_tế	Giá_trị_dự_đoán	
2	Haaland	24	29	32.01512131	53.95985	
3	Salah	32	3	173.9896509	153.0082	
4	Saka	28	13	128.4266441	98.64196	
5	Odegaard	25	15	72.87062472	83.15406	
6	Rice	24	14	22.07608655	65.45353	
7	Saliba	28	7	69.08664113	75.26188	
8	Isak	28	13	71.78483119	98.64196	
9	Mac Allister	21	22	148.6251739	150.4189	
10	Gravenberch	25	27	131.1359196	102.4845	
11	Gvardiol	20	24	178.5704211	143.5572	
12	Haaland	19	29	99.72083578	92.44627	
13	Salah	29	7	32.72290673	73.93005	
14	Saka	23	20	145.5165096	142.8669	
15	Odegaard	19	15	154.5491592	134.0203	
16	Rice	18	12	116.6426675	125.7723	
17	Saliba	29	17	156.4837642	139.1741	
18	Isak	29	14	103.8211633	99.3275	
19	Mac Allister	34	20	109.3192376	123.4864	
20	Gravenberch	27	23	91.23279349	98.53311	
21	Gvardiol	33	25	14.82963408	19.5435	
22	Haaland	32	24	30.49937113	46.91534	
23	Salah	32	27	15.97154528	46.71987	

Analyze feature importance scores for each indicator:





Analysis of Charts

Scatter Plot (Actual Value vs. Predicted Value)

Features:

X-axis: Actual value (million €).

Y-axis: Predicted value (million €).

Red diagonal line: Ideal line (predicted = actual), representing perfect agreement between predicted and actual values.

Scattered points: Each point represents a player, with coordinates (actual value, predicted value).

Metrics:

$R^2 = 0.7836$: Coefficient of determination, measuring the model's explanatory power.

MSE = 668.36 (million €²): Mean Squared Error, measuring the average deviation between predictions and reality.

Comments:

Distribution of Points:

The scattered points are relatively close to the red diagonal line, especially in the range of 0 to 100 million €, indicating the model is quite accurate within this range.

At higher values (above 100 million €), the points tend to scatter further from the diagonal line, showing that the model has a larger error for high-value players.

Some points are very far from the diagonal line (e.g., prediction of 175 million € but actual value of only about 50 million €), indicating the model sometimes overestimates or underestimates significantly.

R^2 Coefficient = 0.7836:

R^2 ranges from 0 to 1, with higher values indicating the model explains the variation in the data better.

$R^2 = 0.7836$ means that approximately 78.36% of the variance in the actual value is explained by the model. This is a good result, indicating the model has reliable predictive power, but there is still approximately 21.64% of the variance that is unexplained (possibly due to factors not included in the data, such as market conditions, personal brand, or external factors).

PROBLEM 4

Mean Squared Error (MSE) = 668.36 (million €²):

MSE measures the average squared error between predicted and actual values.

To better understand this, we can calculate RMSE (Root Mean Squared Error) by taking the square root of MSE:

RMSE = √668.36 ≈ 25.85 million €

RMSE ≈ 25.85 million € shows that, on average, the model's predictions deviate by approximately 25.85 million € from the actual value. With a value range from 0 to 175 million €, this level of error is considerable, especially for high-value players, where the error can be a large proportion of the actual value.

Significance:

The model performs well in predicting player value in the mid-range and low-range segments (0–100 million €) but is less accurate in the high-range segment (above 100 million €). This may be because the training data has fewer samples in the high-range segment, or the features are insufficient to capture the factors influencing the value of superstars.

Large errors in some points (e.g., prediction of 175 million € but actual value of only 50 million €) may indicate outliers or features that have not been optimized (e.g., not accounting for personal brand, market conditions, or external factors such as injuries).

Distribution of Residuals (Error Distribution)

Features:

X-axis: Error (million €), calculated as predicted value minus actual value.

Y-axis: Frequency (number of players with errors in that range).

Red vertical line: Error = 0, ideally, the errors are clustered around this line.

Histogram: Distribution of errors, with columns representing the number of players with errors within each interval.

Comments:

Shape of the distribution:

The error distribution has a shape similar to a normal distribution (bell curve), concentrated around 0, indicating that the model does not have a systematic bias. If the distribution were skewed significantly to the left or right, that would suggest the model consistently undervalues or overvalues.

The errors range from approximately -40 million € to +60 million €, with the majority of errors ranging from -20 million € to +20 million €.

The highest peak is near 0 error, showing many predictions are fairly accurate (small error).

Large Errors:

Some large errors (e.g., +60 million € or -40 million €) suggest that there are some cases where the model predicts quite far off, consistent with the scattered points away from the diagonal line in the above chart.

However, the frequency of large errors (above 40 million € or below -40 million €) is quite low, indicating large prediction errors are not common.

Significance:

The balanced error distribution around 0 is a good sign, showing the model does not tend to overvalue or undervalue systematically.

The range of errors from -40 million € to +60 million € is consistent with RMSE ≈ 25.85 million €, confirming that the average error is within this range.

The large errors (outside the range of ±20 million €) may be related to high-value players or outliers where the current features are insufficient to predict accurately.

What the Charts Reveal About the Model

Overall Performance:

With $R^2 = 0.7836$, the model explains approximately 78.36% of the variance in player value, which is a favorable result. However, there is still approximately 21.64% of the variance that remains unexplained, potentially due to the lack of key features (e.g., personal brand, contract length, or market trends).

RMSE ≈ 25.85 million € indicates a fairly large average error, especially compared to players with low values (under 50 million €). This indicates that the model needs to improve its accuracy, particularly in the high-value segment.

Predictive Power by Segment:

The model performs better in the mid-range and low-range segments (0–100 million €), where the scattered points lie close to the diagonal line.

In the high-range segment (above 100 million €), the model has a larger error, which may be due to a lack of training data for high-value players or the features being insufficient to predict the value of superstars accurately.

Bias and Error:

The error distribution, balanced around 0, indicates the model does not tend to overvalue or undervalue systematically, which is a positive sign.

However, the presence of large errors (up to ±60 million €) shows that the model sometimes predicts inaccurately, especially for high-value players or outliers.

Features and Data:

The current features (such as rating, Position, Nation, Team) appear to be sufficient to predict player value reasonably well, but not enough to predict accurately for special cases (e.g., superstars or players with highly volatile values).

The handling of the rating column (average between current and potential value) can help the model balance between current performance and potential, but other features are still needed to improve accuracy.