Analysis of Python Code for Statistical Analysis of Premier League Player Data (Assignment 1, Part II)

May 5, 2025

Abstract

This report provides a detailed analysis of a Python script designed to fulfill Part II of Assignment 1, which involves statistical analysis of Premier League player data from the 2024-2025 season. The script processes data from a CSV file generated in Part I, performing tasks such as identifying top and bottom performers, calculating statistical summaries, generating histograms, and determining the best-performing team. Utilizing libraries like pandas, NumPy, and Matplotlib, the script ensures robust data handling and visualization. This report explains the script's functionality, structure, and outputs, aligning with the assignment requirements while avoiding personal information.

1 Introduction

The Python script addresses Part II of Assignment 1, which requires advanced statistical analysis of Premier League player data collected in Part I. The input data, stored in `results.csv`, contains detailed statistics for players with over 90 minutes of playtime in the 2024-2025 season, sourced from FBref.com. The script performs five key tasks: identifying the top and bottom three players for each statistic, calculating median, mean, and standard deviation for each statistic across all players and teams, plotting histograms for statistic distributions, identifying teams with the highest average scores per statistic, and analyzing the best-performing team.

This report dissects the scripts structure, detailing its libraries, data processing, task-specific implementations, and output generation. It ensures compliance with the assignments requirements, such as saving results to specified files (`top_3.txt`, `results2.csv`) and generating plots in a designated directory.

1.1 Objectives

The script aims to:

- Identify the top three highest and lowest players for each statistic and save to `top_3.txt`.
- Compute median, mean, and standard deviation for each statistic across all players and by team, saving to `results2.csv`.

- Generate histograms showing the distribution of each statistic for all players and each team, saved in the `plots` directory.
- Identify the team with the highest average score for each statistic, appending results to `top_3.txt`.
- Analyze which team performs best based on overall statistical performance.

2 Libraries Utilized

The script relies on the following Python libraries:

- **pandas**: For data manipulation, including reading CSV files, grouping, and statistical calculations.
- **numpy**: Supports numerical operations and handling of missing data.
- **matplotlib.pyplot**: Generates histograms for visualizing statistic distributions.
- **os**: Manages file paths and directory creation for output files.
- **re**: Uses regular expressions to sanitize filenames for plot saving.

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

Listing 1: Library Imports

3 Initial Setup

The script defines paths for input and output files, ensuring proper organization:

```
results_csv_path = r'C:\Users\nguye\Downloads\results.csv'
top_3_path = r'C:\Users\nguye\Downloads\top_3.txt'
results2_csv_path = r'C:\Users\nguye\Downloads\results2.csv'
plots_dir = r'C:\Users\nguye\Downloads\plots'
os.makedirs(plots_dir, exist_ok=True)
```

Listing 2: File Path Definitions

The `plots` directory is created if it does not exist, ensuring plots are saved correctly.

The script loads the input CSV into a pandas DataFrame and defines a list of valid Premier League teams for the 2024-2025 season to clean the `Squad` column:

```
valid_teams = [
    'Arsenal', 'Aston Villa', 'Bournemouth', ..., 'Wolves'
]
def clean_squad(squad):
    squad = squad.strip()
```

```
    for team in valid_teams:
        if team in squad:
            return team
    return squad
df['Squad'] = df['Squad'].apply(clean_squad)
```

Listing 3: Team List and Squad Cleaning

This ensures team names are standardized, handling cases where players may have played for multiple teams.

The script lists all statistics to analyze, matching the columns from `results.csv`:

```
stats = [
    'MP', 'Starts', 'Min', 'Gls', 'Ast', 'CrdY', 'CrdR', 'xG', '
       npxG', 'xAG',
    'PrgC_standard', 'PrgP_standard', 'PrgR_standard', 'Gls_per90
       ', ..., 'Won%'
]
```

Listing 4: Statistics List

All statistics are converted to numeric types, with non-numeric values coerced to NaN:

```
for stat in stats:
    df[stat] = pd.to_numeric(df[stat], errors='coerce')
```

Listing 5: Numeric Conversion

The script checks for non-numeric columns and prints warnings if any conversions fail.

To account for varying team sizes, cumulative statistics (e.g., `Gls`, `Min`) are normalized by the number of players per team, excluding rate-based statistics (e.g., `Gls_per90`, `Save%`):

```
player_counts = df.groupby('Squad')['Player'].count()
numeric_stats = [stat for stat in stats if stat not in ['
   Gls_per90', 'Ast_per90', ...] ...]
for stat in numeric_stats:
    df[stat] = df.apply(lambda x: x[stat] / player_counts[x['
       Squad']] if pd.notna(x[stat]) else x[stat], axis=1)
```

Listing 6: Team-Level Normalization

Teams with fewer than five players are excluded, and invalid squad names are flagged:

```
teams = df['Squad'].value_counts()
valid_teams = [team for team in teams.index if teams[team] >= 5
   and team in valid_teams]
df = df[df['Squad'].isin(valid_teams)]
```

Listing 7: Team Filtering

4 Task Implementations

4.1 Task 1: Top 3 Highest and Lowest Players

The script identifies the top three highest and lowest players for each statistic, skipping statistics with insufficient data (fewer than three valid entries):

```
top_3_content = []
for stat in stats:
    stat_df = df[['Player', stat]].dropna()
    if len(stat_df) < 3:
        top_3_content.append(f"Top 3 highest for {stat}:
           Insufficient data\n")
        top_3_content.append(f"Top 3 lowest for {stat}:
           Insufficient data\n")
        continue
    top_high = stat_df.nlargest(3, stat)
    top_3_content.append(f"Top 3 highest for {stat}:\n")
    for _, row in top_high.iterrows():
        top_3_content.append(f"{row['Player']}: {row[stat]:.2f}\n
           ")
    top_low = stat_df.nsmallest(3, stat)
    ...
```

Listing 8: Top 3 Players

Results are saved to `top_3.txt` with UTF-8 encoding.

4.2 Task 2: Median, Mean, and Standard Deviation

The script calculates the median, mean, and standard deviation for each statistic across all players and by team:

```
stats_summary = []
for stat in stats:
    valid_data = df[stat].dropna()
    median_val = valid_data.median() if len(valid_data) > 0 else
       'N/a'
    mean_val = valid_data.mean() if len(valid_data) > 0 else 'N/a
       '
    std_val = valid_data.std() if len(valid_data) > 0 else 'N/a'
    row = {'Team': 'all', f'Median of {stat}': median_val, ...}
    stats_summary.append(row)
for team in valid_teams:
    team_df = df[df['Squad'] == team]
    row = {'Team': team}
    for stat in stats:
        valid_data = team_df[stat].dropna()
        row[f'Median of {stat}'] = valid_data.median() if len(
           valid_data) > 0 else 'N/a'
        ...
```

Listing 9: Statistical Summaries

The results are saved to `results2.csv` in the specified format:

4.3 Task 3: Histogram Plots

Histograms are generated for each statistics distribution across all players and by team, using Matplotlib. Statistics with fewer than five valid entries are skipped:

Table 1: Results2.csv Format

Team	Median of Attr 1	Mean of Attr 1	Std of Attr 1	...
all	...	...	...	...
Team 1	...	...	...	...
...	...	...	...	...

```
for stat in stats:
    valid_data = df[stat].dropna()
    if len(valid_data) < 5:
        print(f"Skipping histogram for {stat}: insufficient data
           ")
        continue
    plt.figure(figsize=(10, 6))
    plt.hist(valid_data, bins=30, edgecolor='black')
    plt.title(f'Distribution of {stat} for All Players')
    plt.savefig(os.path.join(plots_dir, f'{sanitize_filename(stat
       )}_all_players.png'))
    plt.close()
```

Listing 10: Histogram Generation

For team-specific histograms, goalkeeper statistics (e.g., `GA90`) are filtered for `Pos == 'GK'`:

```
goalkeeper_stats = ['GA90', 'Save%', 'CS%', 'PK Save%']
for team in valid_teams:
    team_df = df[df['Squad'] == team]
    for stat in stats:
        if stat in goalkeeper_stats:
            valid_data = team_df[team_df['Pos'] == 'GK'][stat].
               dropna()
        else:
            valid_data = team_df[stat].dropna()
        if len(valid_data) < 5:
            continue
        plt.hist(valid_data, bins=30, edgecolor='black')
        plt.title(f'Distribution of {stat} for {team}')
        plt.savefig(os.path.join(plots_dir, f'{sanitize_filename(
           stat)}_{sanitize_filename(team)}.png'))
        plt.close()
```

Listing 11: Team-Specific Histograms

Filenames are sanitized to avoid invalid characters.

4.4 Task 4: Teams with Highest Scores

The script calculates the mean of each statistic by team and identifies the team with the highest average:

```
team_means = df.groupby('Squad')[stats].mean()
top_teams = {}
for stat in stats:
    valid_means = team_means[stat].dropna()
    if len(valid_means) > 0:
        top_team = valid_means.idxmax()
        top_score = valid_means.max()
        top_teams[stat] = (top_team, top_score)
    else:
        top_teams[stat] = ('N/a', 'N/a')
```

Listing 12: Top Teams

Results are appended to `top_3.txt`.

4.5 Task 5: Best-Performing Team Analysis

The script determines the best-performing team by calculating the average of all statistic means per team:

```
team_scores = team_means.mean(axis=1, skipna=True)
valid_scores = team_scores.dropna()
if len(valid_scores) > 0:
    best_team = valid_scores.idxmax()
    best_team_score = valid_scores.max()
else:
    best_team = 'N/a'
    best_team_score = 'N/a'
```

Listing 13: Best Team Analysis

It analyzes the teams performance across offensive, defensive, and possession metrics:

```
offensive = [stat for stat, (team, _) in top_teams.items() if
   team == best_team and stat in ['Gls', 'Ast', 'xG', 'xAG']]
defensive = [stat for stat, (team, _) in top_teams.items() if
   team == best_team and stat in ['Tkl', 'Int', 'Blocks']]
possession = [stat for stat, (team, _) in top_teams.items() if
   team == best_team and stat in ['Touches', 'Carries', '
   PrgC_possession']]
```

Listing 14: Performance Breakdown

The analysis is printed to the console, highlighting the teams strengths.

5 Outputs

The script produces:

- **top_3.txt**: Lists top three highest and lowest players and top teams for each statistic.
- **results2.csv**: Contains median, mean, and standard deviation for each statistic, for all players and by team.

- **Plots**: Histograms in the `plots` directory, named with sanitized statistic and team names (e.g., `Gls_all_players.png`, `Gls_Arsenal.png`).
- **Console Output**: Analysis of the best-performing team, including its strengths in offensive, defensive, and possession metrics.

6 Conclusion and Potential Improvements

The script successfully fulfills Part II of Assignment 1, providing a robust solution for analyzing Premier League player statistics. It handles data cleaning, statistical calculations, visualization, and team performance analysis with clear outputs as required. The use of pandas ensures efficient data processing, while Matplotlib delivers insightful visualizations.

6.1 Potential Improvements

- **Data Validation**: Add checks to ensure input CSV matches expected columns and data types.
- **Plot Customization**: Enhance histograms with log scales for skewed distributions or add team comparisons in a single plot.
- **Error Logging**: Implement detailed logging for debugging and tracking skipped statistics or teams.
- **Performance Optimization**: Use vectorized operations instead of `apply` for normalization to improve speed.
- **Flexible Outputs**: Allow users to specify output formats (e.g., JSON, Excel) or plot styles via parameters.

7 References

- FBref.com About Page
- pandas Documentation
- Matplotlib Histogram Documentation