A Detailed Analysis of Python Code for Scraping Premier League Player Statistics from FBref.com

May 4, 2025

Abstract

This report provides a comprehensive analysis of a Python script designed to scrape detailed player statistics from FBref.com for the Premier League. The script leverages libraries such as Selenium, BeautifulSoup, and pandas to extract, process, and consolidate data from multiple statistical tables into a single CSV file. The code is structured to handle various data types, including standard player metrics, goalkeeping, shooting, passing, and more, while ensuring robust error handling and data consistency. This analysis explains the script's functionality, structure, and key components, offering insights into its design and potential improvements.

1 Introduction

The provided Python script automates the collection of comprehensive player statistics from FBref.com, a reputable source for football analytics. The script targets the Premier League, extracting data from multiple tables, including standard statistics (e.g., goals, assists), goalkeeping metrics, shooting, passing, goal creation actions, defensive actions, possession, and miscellaneous statistics. Using a combination of web scraping (Selenium and BeautifulSoup) and data processing (pandas), the script produces a unified CSV file containing cleaned and aggregated player data, filtered to include only players with over 90 minutes of playtime.

This report dissects the scripts structure, detailing its libraries, configuration, core functions, data processing steps, and output generation. It also discusses potential enhancements to improve efficiency and robustness. The analysis avoids including personal information, such as user names, as per the request.

1.1 Objectives

The script aims to:

- Automate the extraction of player statistics from multiple FBref.com tables.
- Process and clean data, handling inconsistencies and missing values.
- Merge data from different tables using a unique player ID.
- Filter and aggregate data for meaningful analysis.
- Output a structured CSV file for further use.

2 Libraries Utilized

The script relies on several Python libraries, each serving a specific purpose in the data scraping and processing pipeline:

- **pandas**: Facilitates data manipulation, including reading HTML tables, merging DataFrames, and performing aggregations.
- **re**: Uses regular expressions to extract player IDs from HTML links.
- **StringIO**: Enables reading HTML table data into pandas DataFrames.
- **selenium**: Automates browser interactions to load dynamic web content from FBref.com.
- **BeautifulSoup**: Parses HTML to extract specific elements, such as player IDs and names.
- **time**: Introduces delays to prevent server overload during scraping.
- **os**: Manages file paths and directories for saving the output CSV.

```
import pandas as pd
import re
from io import StringIO
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from bs4 import BeautifulSoup
import time
import os
```

Listing 1: Library Imports

3 Configuration

The script begins by defining HTTP headers to mimic a browser, ensuring compatibility with FBref.coms server:

```
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
   x64) AppleWebKit/537.36'}
```

Listing 2: Browser Headers

The output directory and CSV filename are specified, with the directory set to the current working directory for simplicity:

```
output_dir = '.'  % Current directory
csv_filename = 'results.csv'  % Required filename
```

Listing 3: Output Configuration

A dictionary, `table_configs`, defines the structure for scraping eight distinct tables from FBref.com. Each table configuration includes:

- `url_type`: The URL segment for the table (e.g., `stats`, `keepers`).
- `columns`: The list of columns to extract from the table.
- `rename`: A dictionary mapping raw column names to standardized names for consistency.

Table 1: Table Configurations

Table Type	Description
Standard	General metrics (e.g., matches played, goals, assists, xG)
Goalkeeping	Goalkeeper-specific metrics (e.g., goals against, save percentage)
Shooting	Shooting statistics (e.g., shots on target, goal per shot)
Passing	Passing metrics (e.g., completion percentage, key passes)
Goal Creation (GCA)	Goal and shot-creating actions
Defense	Defensive actions (e.g., tackles, interceptions)
Possession	Possession metrics (e.g., touches, carries)
Miscellaneous	Other metrics (e.g., fouls, recoveries)

For example, the standard table configuration extracts columns like `Player`, `Nation`, `Gls`, and `xG`, renaming metrics like `Gls.1` to `Gls_per90` for clarity:

```
'standard': {
    'url_type': 'stats',
    'columns': ['Player', 'Nation', 'Pos', 'Squad', 'Age', 'MP',
       'Starts', 'Min', 'Gls', 'Ast', ...],
    'rename': {
        'Nation': 'Nation', 'Pos': 'Pos', 'Squad': 'Squad', 'Age
           ': 'Age', ...,
        'Gls.1': 'Gls_per90', 'Ast.1': 'Ast_per90', ...
    }
}
```

Listing 4: Standard Table Configuration

4 Core Functionality: `scrape_table`

The `scrape_table` function is the backbone of the script, responsible for scraping a single table based on its configuration. It returns a processed pandas DataFrame or `None` if the table cannot be scraped.

4.1 Key Steps

1. **URL Construction**: Builds the table-specific URL (e.g., https://fbref.com/en/comps/9/stats/Premier-League-Stats).
2. **Browser Setup**: Initializes a headless Chrome browser using Selenium with the specified user-agent.

3. **Page Navigation**: Loads the page and waits 3 seconds for content to render.
4. **Table Extraction**: Locates the player stats table by ID (e.g., `stats_stats_9`) or CSS class (`stats_table`).
5. **HTML Parsing**: Uses BeautifulSoup to parse the tables HTML.
6. **Player ID Extraction**: Extracts player IDs and names from links in the `Player` column using regular expressions.
7. **DataFrame Creation**: Reads the table into a pandas DataFrame.
8. **Column Processing**: Flattens multi-level columns, disambiguates duplicates (e.g., `Gls_total` vs. `Gls_per90`), and renames columns per the configuration.
9. **Data Cleaning**: Removes duplicate columns and handles missing data.
10. **Player ID Integration**: Adds a `Player_ID` column to the DataFrame.
11. **Column Selection**: Selects required columns, adding `N/a` for missing ones.
12. **Type Conversion**: Converts numeric columns to appropriate types, handling errors gracefully.
13. **Return**: Returns the processed DataFrame.

```
def scrape_table(config):
    """
    Scrapes a specified table from FBref.com using Selenium,
       extracting player IDs and required columns.

    Args:
        config (dict): Configuration with URL type, columns, and
           renaming rules.

    Returns:
        pd.DataFrame: Processed DataFrame with selected and
           renamed columns, or None if table not found.
    """
```

Listing 5: `scrape_table` Function

The function includes robust error handling, such as:

- Checking for table existence and returning `None` if not found.
- Handling HTML parsing errors during DataFrame creation.
- Managing column mismatches and duplicate columns.
- Ensuring proper browser cleanup with a `finally` block.

5 Data Processing and Merging

The script iterates through each table configuration, calling `scrape_table` and storing the resulting DataFrames in a dictionary (`dfs`). Failed tables are tracked in `failed_tables` for reporting.

If the standard table is available, it filters players with over 90 minutes of playtime:

```
if 'standard' in dfs:
    df_standard = dfs['standard']
    df_standard['Min'] = pd.to_numeric(df_standard['Min'], errors
       ='coerce')
```

Listing 6: Filtering Standard Stats

The script merges DataFrames starting with the standard table (or the first available table if standard is missing). Other tables are merged on `Player_ID` using a left join:

```
if df_standard is not None:
    merged_df = df_standard
else:
    merged_df = next(iter(dfs.values()), None)
for table_type in table_configs:
    if table_type != 'standard' and table_type in dfs:
        merge_columns = ['Player_ID'] + list(table_configs[
           table_type]['rename'].values())
        merged_df = pd.merge(merged_df, dfs[table_type][
           available_cols], on='Player_ID', how='left')
```

Listing 7: Merging DataFrames

To handle players who played for multiple teams, the script aggregates data by `Player_ID` using a dictionary of aggregation functions:

- `sum`: For cumulative metrics (e.g., `Gls`, `Min`).
- `mean`: For rates (e.g., `Gls_per90`, `Save%`).
- `first`: For static fields (e.g., `Nation`, `Pos`).
- `lambda`: Joins unique squads with commas for `Squad`.

```
agg_dict = {
    'Player': 'first', 'Nation': 'first', 'Pos': 'first',
    'Squad': lambda x: ', '.join(x.dropna().unique()),
    'Min': 'sum', 'Gls': 'sum', 'Gls_per90': 'mean', ...
}
merged_df = merged_df.groupby('Player_ID').agg(agg_dict).
   reset_index(drop=True)
```

Listing 8: Data Aggregation

The script filters for players with over 90 minutes, sorts by first name, fills missing values with `N/a`, and ensures all specified columns are present:

```
merged_df = merged_df[merged_df['Min'] > 90]
merged_df['First_Name'] = merged_df['Player'].apply(lambda x: x.
    split()[0])
merged_df = merged_df.sort_values(by='First_Name').drop(columns
    =['First_Name'])
merged_df = merged_df.fillna("N/a")
```

Listing 9: Final Data Processing

6 Output Generation

The script defines a comprehensive list of final columns, covering all relevant metrics:

Table 2: Selected Final Columns

Category	**Examples**
Basic Info	Player, Nation, Squad, Pos, Age
Playing Time	MP, Starts, Min
Performance	Gls, Ast, CrdY, CrdR
Expected Metrics	xG, npxG, xAG
Goalkeeping	GA90, Save%, CS%
Shooting	SoT%, SoT/90, G/Sh
Passing	Total Cmp%, KP, PPA
Defense	Tkl, TklW, Int
Possession	Touches, Carries, CPA
Miscellaneous	Fls, Fld, Off, Recov

The final DataFrame is saved to a CSV file in the specified directory with UTF-8 encoding:

```
os.makedirs(output_dir, exist_ok=True)
csv_path = os.path.join(output_dir, csv_filename)
final_df.to_csv(csv_path, index=False, encoding='utf-8-sig')
```

Listing 10: Saving Output

7 Conclusion and Potential Improvements

The script effectively automates the collection, processing, and consolidation of Premier League player statistics from FBref.com. Its modular design, robust error handling, and comprehensive data processing make it a powerful tool for football analytics. The resulting CSV file is well-structured for further analysis, with standardized column names and cleaned data.

7.1 Potential Improvements

- **Dynamic Waiting**: Replace the fixed 3-second delay with Seleniums explicit waits to optimize page loading.

- **Parallel Scraping**: Scrape tables concurrently using multiprocessing, while respecting server limits.
- **Enhanced Error Handling**: Implement retries for failed requests and detailed logging for debugging.
- **Flexible Configuration**: Allow users to specify different leagues or seasons via parameters.
- **Data Validation**: Add checks to verify data consistency (e.g., ensuring `Min` aligns with `MP`).

8 References

- FBref.com About Page
- FBref Premier League Statistics
- Selenium Documentation
- pandas Documentation