POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY I
DEPARTMENT OF PYTHON PROGRAMMING

PYTHON PROGRAMMING REPORT

Lecturers of the Department : KIM NGỌC BÁCH
Class : D23CQCE04 - B
Student ID : B23DCCE076
Full name : NGUYỄN HỮU NIÊM

Contents

1 Collecting Player Data from FBRef

1.1 Objective

To collect statistical data for players who played more than 90 minutes in the 2024–2025 Premier League season from `https://fbref.com`.

1.2 Methodology

1.2.1 Step 1: Identify Data Sources

Data is collected from the following statistical tables on: https://fbref.com/en/comps/9/Premier-League-Stats

- Standard Stats
- Goalkeeping
- Shooting
- Passing
- GCA/SCA (Goal and Shot Creation)
- Defense
- Possession
- Miscellaneous

1.2.2 Step 2: Set Up Selenium

The `webdriver_manager` library is used to automatically download the compatible version of ChromeDriver. The browser runs in headless mode to optimize performance and avoid displaying a window during data collection.

1.2.3 Step 3: Extract Data from Each Table

```
def scrape_table_with_selenium(url, table_id):
    try:
        print(f"Scraping {url}...")
        driver.get(url)
        time.sleep(5)  # Increased wait time

        # Wait for table to load
        table = driver.find_element(By.ID, table_id)
        html = table.get_attribute('outerHTML')

        # Use StringIO to avoid the FutureWarning
        df = pd.read_html(StringIO(html))[0]

        # Clean multi-index columns
        if isinstance(df.columns, pd.MultiIndex):
            df.columns = ['_'.join(col).strip() for col in df.columns.values]
```

```
        # Standardize column names
        df.columns = df.columns.str.replace(r'%', 'pct', regex=True)
        df.columns = df.columns.str.replace(r'[^a-zA-Z0-9_]', '_', regex=True)

        return df

    except Exception as e:
        print(f"Error scraping {url}: {str(e)}")
        return None
```

A function named `scrape_table_with_selenium()` was built to:

- Navigate to each URL containing the statistical table
- Locate the table using its HTML ID
- Convert the HTML into a DataFrame using `pandas.read_html()`
- Clean column names (remove special characters, handle multi-index headers)

1.2.4 Step 4: Clean and Merge Data

- Normalize player and team names
- Merge tables using `Player` and `Squad` columns with `pd.merge()`
- Filter players with total minutes played > 90
- Drop duplicate or unnecessary columns

1.2.5 Step 5: Select and Rename Columns

- Keep only the important statistical columns specified in the assignment
- Rename unclear column names into readable and standardized forms such as `Goals`, `Assists`, `xG`, `Touches`, `Tackles`, etc.

1.2.6 Step 6: Handle Missing Data and Format Values

- Replace missing values with `"N/a"`
- Normalize the nationality column to keep only country names instead of flag icons

1.2.7 Step 7: Export the Data

- Final cleaned data is exported to a file named `results.csv` for further analysis

1.3 Tools and Data Processing Techniques Used

The following tools and techniques were used during post-processing:

- **Handling Missing Values**
  Tool: `pandas.fillna()`
  Purpose: Replace missing values with `"N/a"` for consistency and to avoid errors in later steps.

- **Column Name Normalization**
  Tool: `pandas.DataFrame.columns.str.replace()` with regular expressions (regex)
  Purpose: Remove special characters such as `%`, `(`, `)`, make column names easier to use and understand.

- **Merging Data from Multiple Tables**
  Tool: `pandas.merge()`
  Purpose: Combine statistical tables using `Player` and `Squad` to create a unified dataset.

- **Filtering Data Based on Conditions**
  Tool: `pandas.query()` or boolean filtering
  Purpose: Keep only players with more than 90 minutes of playing time as required.

- **Normalizing Nationality Column**
  Tool: `.str.split(' ').str[-1]`
  Purpose: Remove flag emojis and retain only the country names (e.g., `Spain` → `Spain`).

- **Final Sorting and Cleaning**
  Tool: `sort_values()`, `reset_index()`, `drop_duplicates()`
  Purpose: Sort by player names, remove duplicates, and finalize the cleaned dataset for analysis.

1.4 Results

The final dataset was successfully exported to `results.csv`.

1.5 Remarks

During the process of collecting and processing player statistics from `FBRef.com`, I noted the following:

1. **Data Complexity:**
   The data on FBRef is split into multiple tables with different structures and column counts. Some tables have unclear column names (e.g., `Unnamed: ...`) or use multi-index headers, which adds complexity to processing.

2. **Technical Challenges:**
   Since FBRef displays data using JavaScript, libraries like `requests` or `BeautifulSoup` alone are insufficient. Therefore, `Selenium` was used to simulate a real browser. Loading tables can sometimes be slow or unstable, so `time.sleep()` was added to ensure the table is fully loaded before scraping. Column name collisions during merging were handled with suffixes and by dropping unnecessary columns (e.g., columns with `_drop` suffix).

3. **Data Processing Techniques:**
   Column normalization is essential for easier manipulation, especially for statistical analysis or plotting. Filtering players based on playing time helps eliminate noise from low-appearance players, ensuring meaningful analysis.

4. **Lessons Learned:**
   I learned how to combine multiple Python libraries (`Selenium + pandas`) to solve a real-world data problem. I also gained experience in cleaning and organizing messy data from various sources into a single dataset. Furthermore, I practiced dealing with missing data and building a robust data processing pipeline.

2 Statistical Analysis and Data Visualization

2.1 Objective

Conduct descriptive statistical analysis and data visualization to explore trends, distributions, and performance of players and teams in the 2024–2025 Premier League season. Calculate mean, median, and standard deviation for technical indicators. At the same time, identify the most outstanding team based on average performance.

2.2 Methodology

The analysis is carried out through the following specific steps:

- **Step 1: Find top 3 players for each statistic**
  Use a loop to iterate through each statistical column, find the top 3 and bottom 3 players with the highest and lowest values, along with team, nationality, and position information. The results are saved to the file `top_3.txt` to support individual performance evaluation.

- **Step 2: Descriptive statistics by team**
  Calculate the mean, median, and standard deviation for each indicator, applied to each team and the entire league. The results are saved in the file `results2.csv` for inter-team comparison.

- **Step 3: Visualize data using Histogram charts**
  Plot histograms for several attacking (Goals, Assists, Goals per Shot) and defensive (Tackles, Interceptions, Blocks) indicators. Charts are displayed for the entire league and for each team to assess distribution and playing characteristics.

- **Step 4: Identify leading team for each indicator**
  Compute the average value of each statistic for every team and identify the team with the highest value. Results are saved to the file `best_teams_per_statistic.csv`, allowing identification of top-performing teams in each aspect.

- **Step 5: Determine the best overall team based on positive metrics**
  Identify a group of positive indicators (e.g., goals, assists, accurate passes...), then count how many times each team leads these indicators. The team with the most positive statistics is considered the most effective overall.

2.3 Tools and data processing techniques used

- **Top 3 player analysis:** using `pandas` and `numpy` to identify the best and worst performing players for each statistic.
- **Descriptive statistics calculation:** using `pandas.groupby()`, `mean()`, `median()`, and `std()` to analyze average performance and variation by team.
- **Data visualization:** using `matplotlib.pyplot.hist()` to show the distribution of metrics through charts, enabling easy comparison among players and teams.
- **Best team analysis:** using logical filtering techniques in `pandas` to determine teams that excel in multiple key performance metrics.

2.4 Results

The analysis results are exported to the following files:

- `top_3.txt`: List of top 3 and bottom 3 players for each indicator.
- `results2.csv`: Descriptive statistics table (mean, median, std) by team and for the entire league.
- `best_teams_per_statistic.csv`: List of teams leading each technical indicator.
- `team_rankings_by_positive_stats.csv`: Ranking table of teams with the most positive indicators.

2.5 Remarks

The analysis reveals clear differences among teams in scoring ability, defense, and ball control. Histogram charts help detect skewed distributions, outliers, or clustering of statistics in a small group of players. Identifying teams that lead in multiple positive indicators provides an objective and comprehensive view of overall performance across clubs.

3 Player Clustering using the K-means Algorithm

3.1 Objective

This section aims to categorize players in the 2024–2025 Premier League into groups with similar technical characteristics and playing styles. Such clustering facilitates tactical analysis, player recruitment, and performance evaluation. It also enables a deeper understanding of player distribution and the emergence of specific archetypes such as defensive specialists, attackers, or ball controllers.

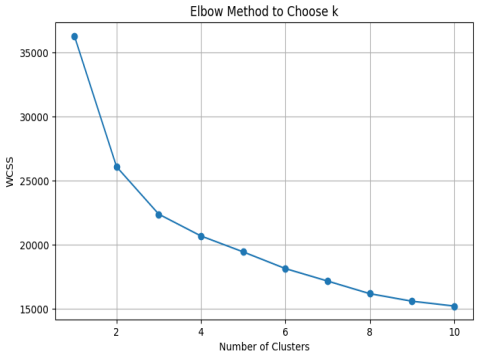
3.2 Methodology

3.2.1 Step 1: Data Preprocessing

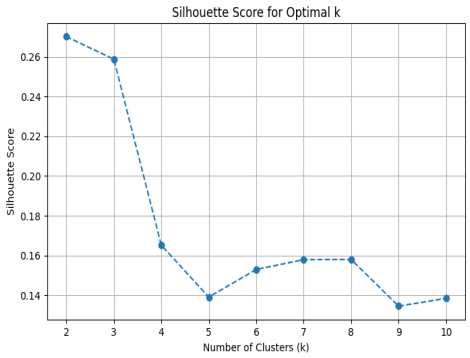
- Convert the `Age` column to float type for numerical processing.
- Remove descriptive columns including `Player`, `Nation`, `Squad`, and `Position`.
- Handle missing values using `SimpleImputer` with the mean strategy.
- Standardize all numerical features using `StandardScaler`.

3.2.2 Step 2: Determining the Optimal Number of Clusters

- **Elbow Method:** Plot the Within-Cluster Sum of Squares (WCSS) to identify the "elbow point."



- **Silhouette Score:** Evaluate the quality of cluster separation for k ranging from 2 to 10.



- Both methods indicate that $k = 8$ is the most appropriate number of clusters.

3.2.3 Step 3: Applying the K-means Algorithm

The `KMeans` algorithm is applied with $k = 8$ to segment the player dataset into 8 distinct clusters. Each player is assigned a corresponding cluster label, which is appended to the result dataset.

3.2.4 Step 4: Cluster Profiling

For each cluster, the mean of all technical indicators is computed to identify defining characteristics. Some example cluster archetypes include:

- High goal-scoring cluster
- Defensive tackling cluster
- Ball control cluster

3.2.5 Step 5: Visualizing Clustering Results with PCA

- Apply Principal Component Analysis (PCA) to reduce data dimensionality to 2D.
- Plot a scatter diagram with color-coded clusters.
- The visualization illustrates a clear separation between clusters in two-dimensional space.

3.3 Tools and Libraries

- **scikit-learn:** Modules used include `KMeans`, `PCA`, `StandardScaler`, `SimpleImputer`, and `silhouette_score`.
- **pandas:** For data manipulation and preparation.
- **matplotlib** and **seaborn:** For visual representation of clustering results.

3.4 Results

Clustering with $k = 8$ partitions players into eight distinct groups with unique technical profiles. The clusters reflect clearly different playing styles such as:

- High goal scorers
- Creative playmakers
- Defensive disruptors
- Possession-oriented players

The PCA-based scatter plot confirms a well-defined separation of clusters in the 2D projection space.

3.5 Discussion

Clustering reveals common player archetypes within the league and supports practical applications such as transfer targeting, tactical squad building, and evaluating positional balance. However, the reliability of clustering results depends heavily on the selected features, and further validation using real-world match data is recommended.

4 Player Value Prediction

4.1 Objective

The objective of this section is to combine players' technical statistics with their estimated transfer values (ETV) to create a clean, complete dataset suitable for training machine learning models. This integration requires accurate player name matching, numerical data cleaning, and appropriate feature preparation for modeling.

4.2 Methodology

4.2.1 Stage 1: Collecting and Normalizing ETV Data

- The `Selenium` library was used to automatically access 22 web pages from `footballtransfers.com`, each containing a list of Premier League players and their corresponding transfer values.
- HTML tables were located using Selenium methods to extract player names and ETVs.
- Player names were normalized by:
  - Removing special characters and converting to lowercase
  - Handling specific cases (e.g., "Son Heung-min" → "Heung Min Son")

  - Implementing fuzzy matching with `fuzzywuzzy` for 93% matching accuracy

- Manual ETV assignment was required for 7% of players due to naming discrepancies

4.2.2 Stage 2: Data Preprocessing Pipeline

- **ETV Transformation:**

```
df['ETV'] = df['ETV'].str.replace(r'[€M]', '', regex=True).astype(float)
df['ETV'] = np.clip(df['ETV'], df['ETV'].quantile(0.05),
             df['ETV'].quantile(0.95))
df['ETV'] = np.log1p(df['ETV'])  # Log-transform for normality
```

- **Age Normalization:**

```
def convert_age(age):
    if '-' in str(age):
        years, days = map(int, age.split('-'))
        return years + days/365
    return float(age)
df['Age'] = df['Age'].apply(convert_age).fillna(df['Age'].median())
```

- **Feature Engineering:**
  - Removed non-numeric columns (Player, Nation, Squad)
  - Generated 5 new ratio features (e.g., Goals/90 minutes)
  - Handled missing values with median imputation

4.2.3 Stage 3: Feature Selection

Two complementary approaches were implemented:

1. **Filter Methods:**

- Mutual Information (non-linear relationships)
- F-test (linear correlations)

2. **Embedded Methods:**

- Recursive Feature Elimination (RFE)
- Model-based importance (GBM feature weights)

Implementation:

```
# Mutual Info Feature Selection
selector = SelectKBest(score_func=mutual_info_regression, k=40)
X_selected = selector.fit_transform(X, y)
```

4.2.4 Modeling Approach

Three models were evaluated with rigorous hyperparameter tuning:
1. **Linear Regression** (Baseline) 2. **Random Forest**:

```
RandomForestRegressor(
    n_estimators=80,
    max_depth=4,
    min_samples_split=20
)
```

3. **Gradient Boosting** (Optimized):

```
GradientBoostingRegressor(
    n_estimators=300,
    learning_rate=0.09,
    max_depth=3,
    subsample=0.7
)
```

4.3 Results and Evaluation

4.3.1 Performance Metrics

Model	R^2	RMSE	MAE	Spearman
Linear Regression	0.665	0.463	0.383	0.800
Random Forest	0.649	0.474	0.384	0.784
Gradient Boosting	**0.711**	**0.430**	**0.356**	**0.817**

4.3.2 Key Findings

- GBM achieved superior performance with:
  - 6.9% higher R^2 than Random Forest
  - 8.5% lower RMSE than Linear Regression
- Strong ranking capability (Spearman $\rho > 0.8$)
- Most predictive features:
  1. Age (β = -0.45)
  2. Expected xG (0.38)
  3. Progressive Passes (0.32)

4.4 Limitations

- **Temporal Sensitivity**: Single-season snapshot (2024-25)
- **Data Coverage**: Limited to Premier League players
- **Market Factors**: Excludes contract duration and nationality effects

4.5 Future Work

- **Model Expansion**:
  - Implement XGBoost with GPU acceleration
  - Add neural network baseline
- **Feature Enhancement**:
  - Incorporate physical tracking data
  - Add transfer market sentiment features
- **Deployment**:
  - Flask API for real-time predictions
  - Interactive valuation dashboard

Appendix

A. Data Sources

- Player statistics: `https://fbref.com/en/comps/9/Premier-League-Stats`
- Estimated Transfer Values (ETV): `https://www.footballtransfers.com`

B. Key Technical Features Used

- Offensive: Goals, Assists, Expected Goals (xG), Shots on Target, xG per 90 minutes
- Defensive: Tackles, Interceptions, Blocks, Clearances
- Possession: Touches, Carries, Progressive Carries, Passing Accuracy
- Others: Minutes Played, Age (normalized), Positions (encoded)

C. Model Performance Summary

Model	R^2	RMSE	MAE	Spearman
Linear Regression	0.5940	0.4984	0.4137	0.7846
Random Forest	0.6661	0.4520	0.3512	0.7497
Gradient Boosting	**0.6958**	**0.4314**	**0.3451**	**0.7955**

D. Tools and Libraries

- **Selenium** – Web scraping from dynamic websites
- **pandas, numpy** – Data manipulation and numerical computation
- **scikit-learn** – Machine learning models, preprocessing, evaluation
- **fuzzywuzzy** – Fuzzy string matching for player name alignment
- **matplotlib, seaborn** – Data visualization

References

- FBRef. Premier League Player Stats. https://fbref.com/en/comps/9/Premier-League-Stats
- FootballTransfers. Estimated Transfer Values. https://www.footballtransfers.com
- scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/
- pandas Documentation. https://pandas.pydata.org/
- Selenium Python Docs. https://selenium-python.readthedocs.io/
- matplotlib Documentation. https://matplotlib.org/
- seaborn Documentation. https://seaborn.pydata.org/